Part A

SPOS Lab Assignment 1

Assignment No: 1	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Pass 1 of a two-pass assembler

Problem Statement: Design suitable data structures and implement pass-I of a two-pass assembler. Implementation should consist of a few instructions from each category and few assembler directives.

Theory: Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.

It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. If assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

- Pass 1
- Pass 2

Pass 1 Purpose: Define symbols and literals

- Define length of machine instructions (MOTGET1)
- Keep track of location counter (LC)
- Remember values of symbols until pass 2(STSTO)
- Process some psuedo ops, e.g., EQU, DS(POTGET1)
- Remember literals(LITSTO)

Data structures: **Pass1:** Optab, Symtab, Littab, Pooltab

Symbol table(SYMTAB): It contains entries such as symbol, it's address and value.

Literal table(LITTAB): it contains entries such as literal and it's value.

Pool table(POOLTAB): Contains literal number of the starting literal of each literal pool.

1. Input source program.
2. A location counter(LC), used to keep track of each instruction's location.
3. A table, Machine Operation Table(MOT), that indicates the symbolic mnemonic for each instruction and its length(2, 4 or 6 bytes).
4. A table, Psuedo Operation Table(POT), that indicates the symbolic mnemonic and action to be taken for each psuedo-op in pass 1.
5. A table, the literal table(LT), that is used to store each literal encountered and its corresponding assigned location.
6. A copy of the input to be used later by pass 2. The same may be stored in a secondary storage device or the original source program can be read by the assembler a second time for pass 2.

Format of Data structures :

- Pass 2 requires a MOT containing name, length, binary code and format; pass 1 requires only name and length
- Two separate tables can be used with different formats and contents or same table for both passes
- Same for POT
- Once we decide what information belongs in which database, format specification is necessary
- By format, we mean the format in which symbols are stored and the coding conventions

Program:

input:

```
**    START     2000
**    MOVER     ='5'
**    MOVEM     X
L1    MOVEM     ='2'
**    ORIGIN    L1+3
**    LTORG     **
X     DS          1
**    END         **
```

optab:

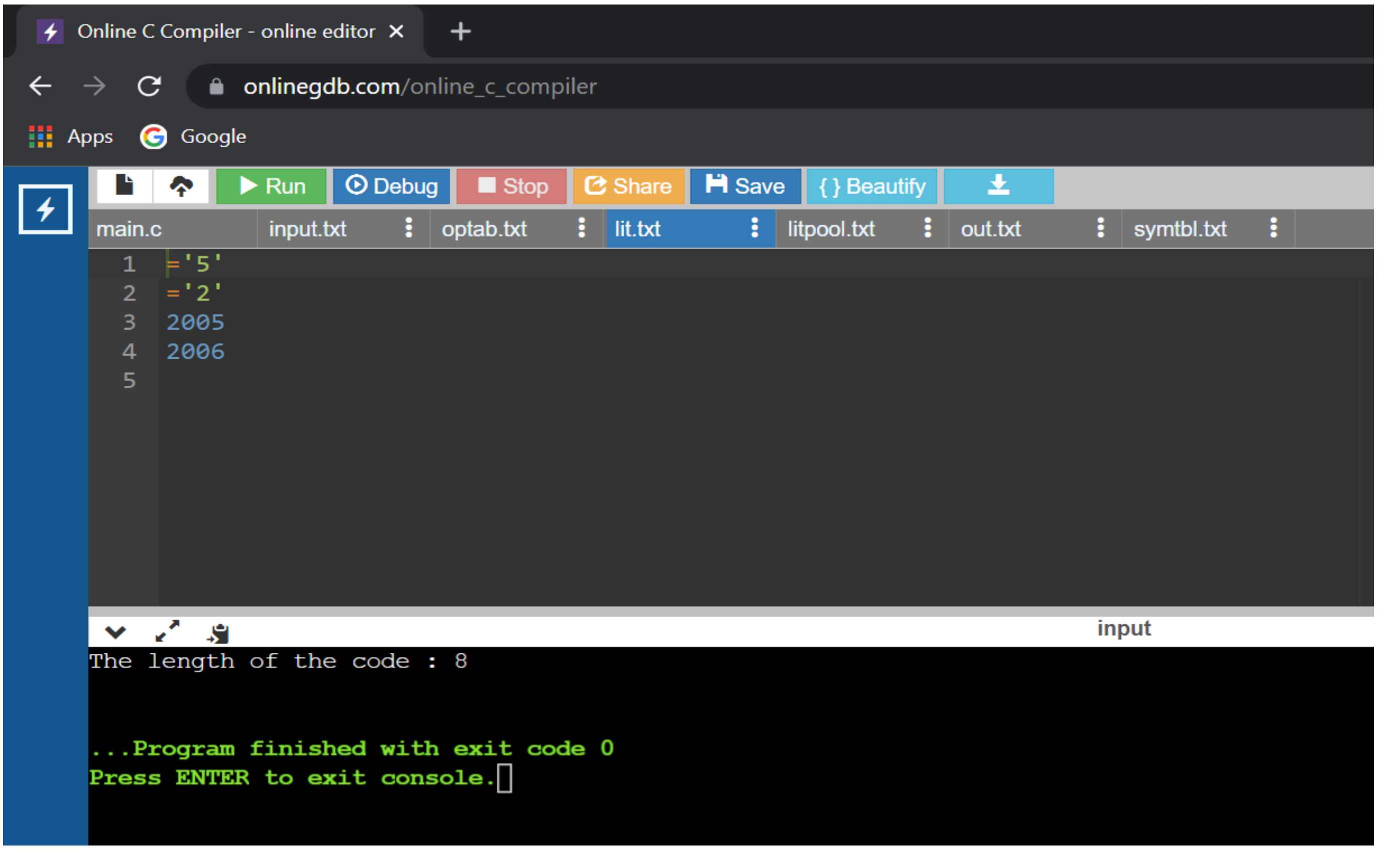
```
START     **
MOVER     04
MOVEM     05
END       **
```

Pass1:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main() {
   int lit,  count=0;
   char opcode[10], operand[10], label[10], code[10], mnemonic[10];
   int locctr, start, length;
   char *tmp;
   FILE *fp1, *fp2, *fp3, *fp4, *fp5, *fp6;
          fp1 = fopen("input.txt", "r");
          fp2 = fopen("optab.txt", "r");
```

```
            fp3 = fopen("symtbl.txt", "w");
            fp4 = fopen("out.txt", "w");
            fp5 = fopen("lit.txt", "w");
            fp6 = fopen("litpool.txt", "w");
fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
fprintf(fp6, "%d\n", count);
      if(strcmp(opcode, "START")==0) {
      start = atoi(operand);
      locctr = start;
      fprintf(fp4, "\t%s\t%s\t%s\n", label, opcode, operand);
      fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
  }
  else {
      locctr = 0;
  }
 while(strcmp(opcode, "END")!=0) {
      fprintf(fp4, "%d\t", locctr);
      tmp = strstr(operand,"=");
      if (tmp != NULL) {
         count++;
         fprintf(fp5, "%s\n", operand);
      }
      if(strcmp(label, "**")!=0) {
         fprintf(fp3, "%s\t%d\n", label, locctr);
      }
      fscanf(fp2, "%s\t%s", code, mnemonic);
         while(strcmp(code, "END")!=0) {
                 if(strcmp(opcode, code)==0) {
                        locctr+=1;
                        break;
                    }
                   fscanf(fp2, "%s\t%s", code, mnemonic);
           }
       if(strcmp(opcode, "ORIGIN")==0) {
           locctr+=3;
       }
       else if(strcmp(opcode, "DS")==0) {
           locctr+=1;
       }
       else if(strcmp(opcode, "LTORG")==0) {
           for (lit=0; lit<count; lit++) {
               fprintf(fp5, "%d\n", locctr + lit);
           }
           locctr+=count;
           fprintf(fp6, "%d\n", count);
       }
   fprintf(fp4, "%s\t%s\t%s\t\n", label, opcode, operand);
   fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
   }
fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);
   length = locctr-start;
   printf("The length of the code : %d\n", length);
            fclose(fp1);
            fclose(fp2);
            fclose(fp3);
            fclose(fp4); }
```

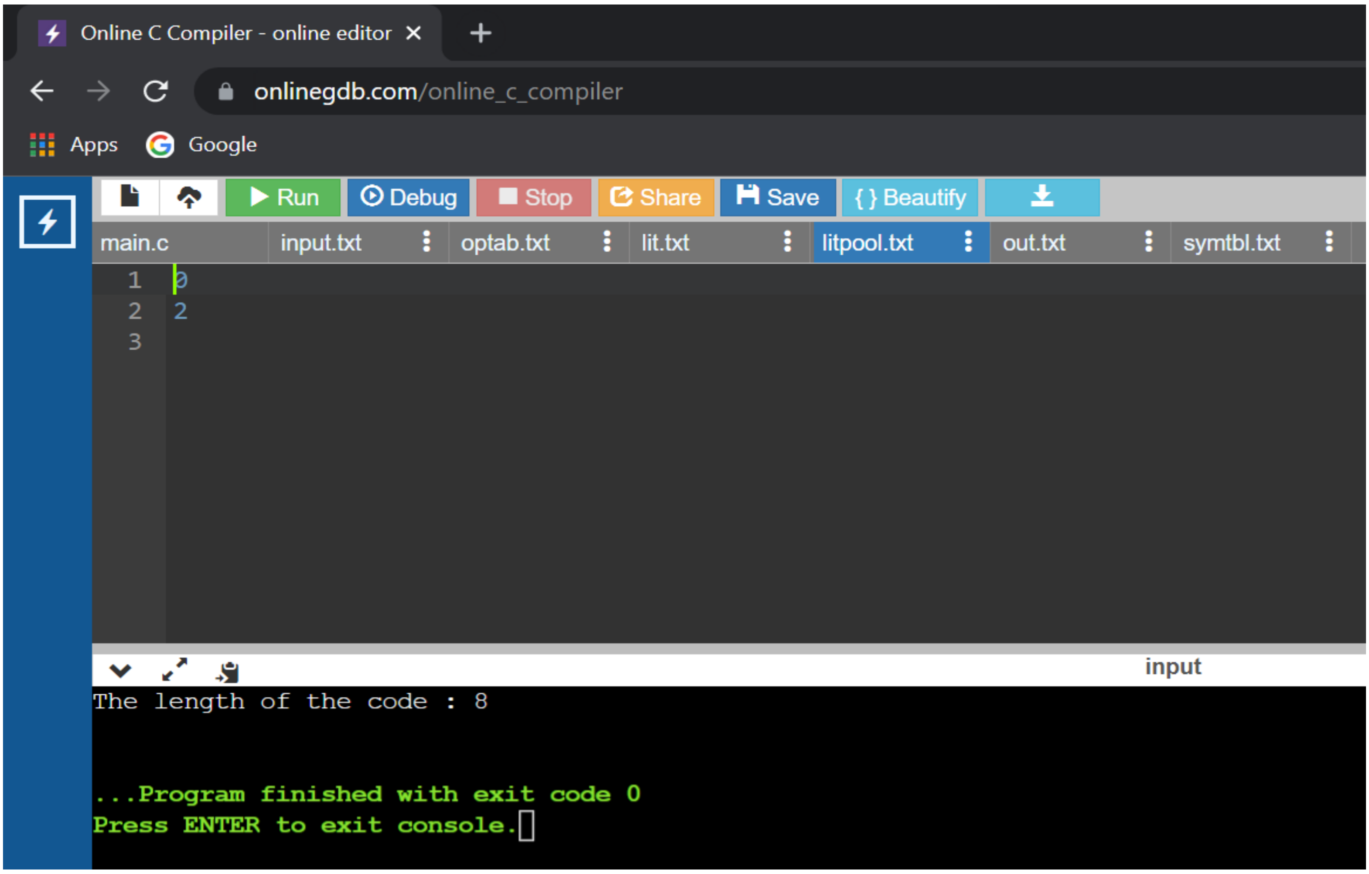
Output: ***lit.txt :***



```
='5'
='2'
2005
2006
```

```
The length of the code : 8


...Program finished with exit code 0
Press ENTER to exit console.
```

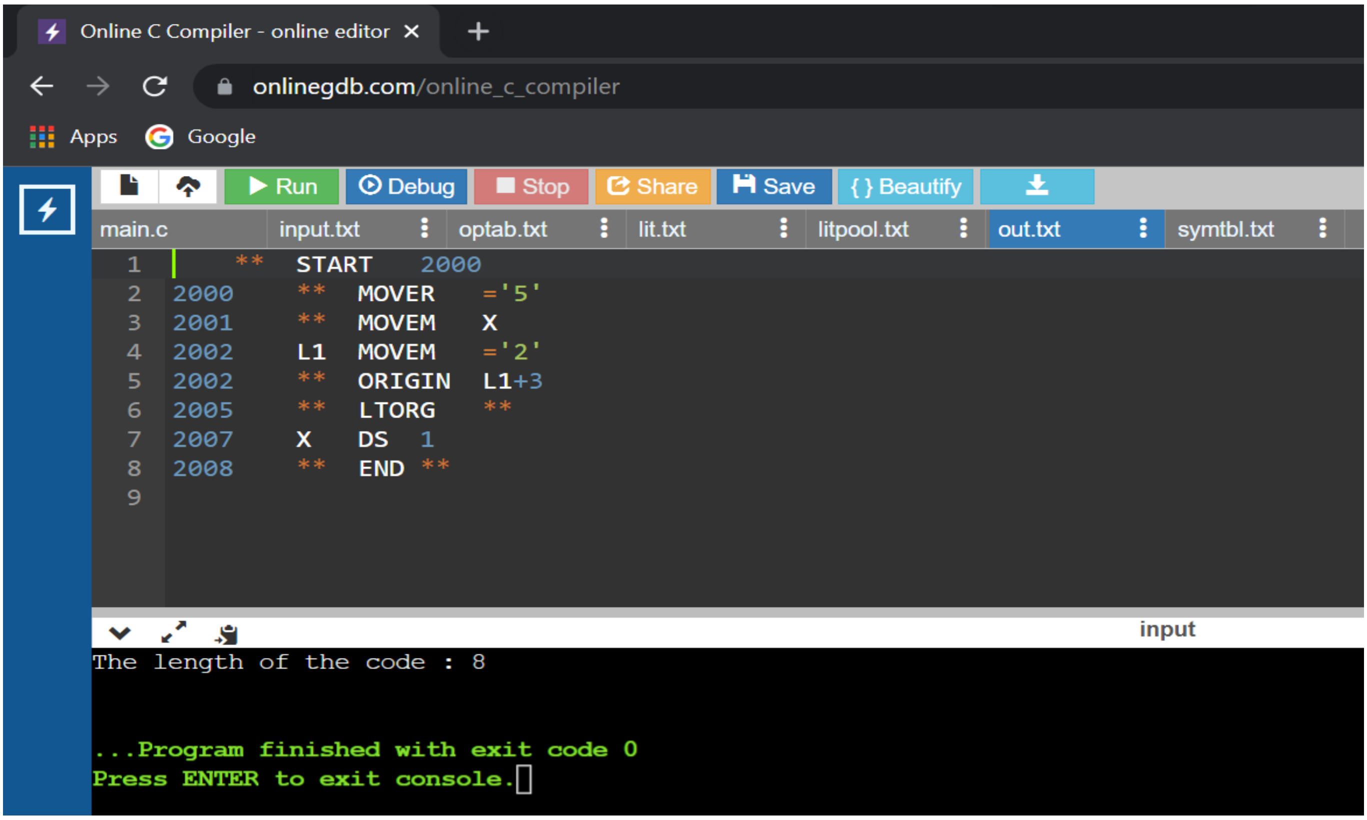
litpool.txt :



```
0
2
```

```
The length of the code : 8


...Program finished with exit code 0
Press ENTER to exit console.
```

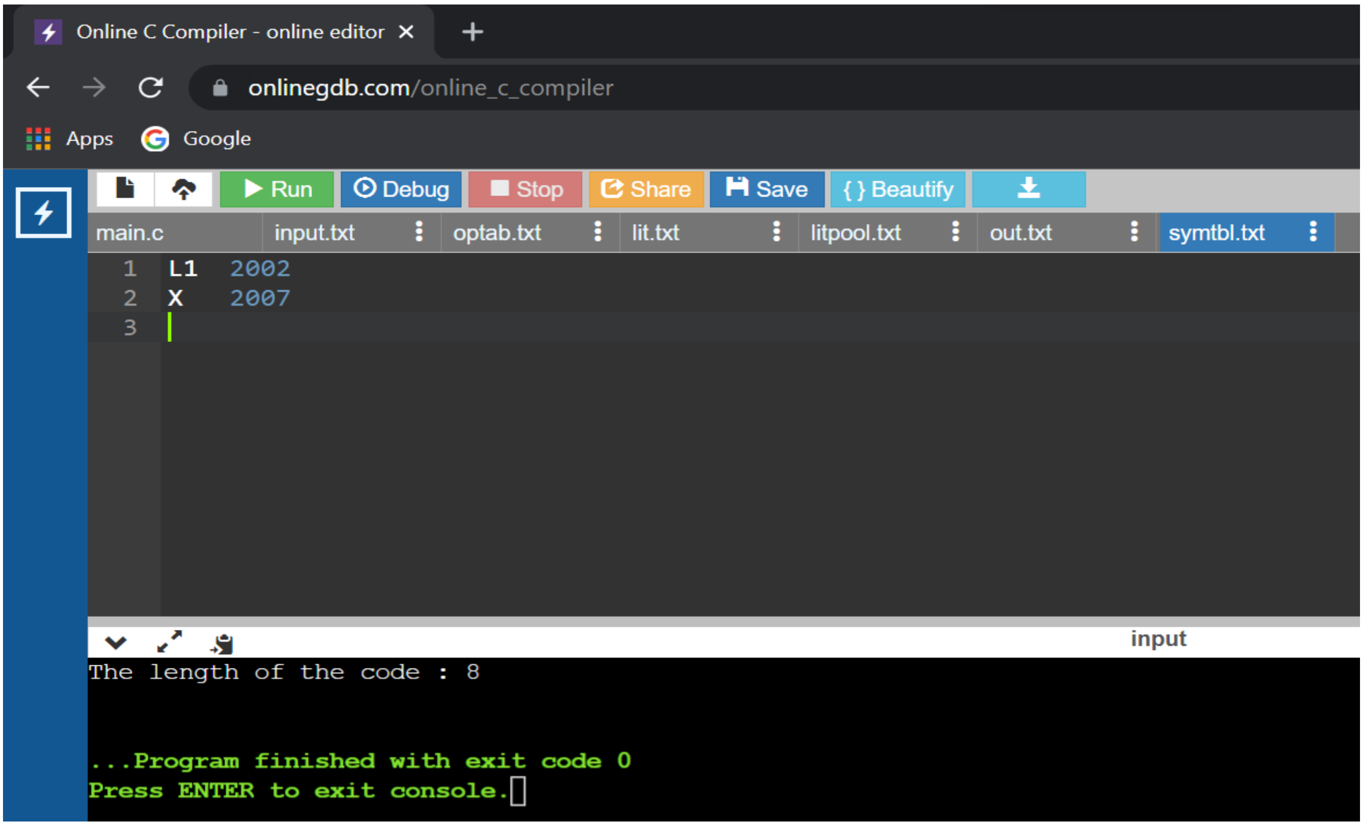
out.txt :



```
        **      START   2000
2000    **      MOVER   ='5'
2001    **      MOVEM   X
2002    L1      MOVEM   ='2'
2002    **      ORIGIN  L1+3
2005    **      LTORG   **
2007    X       DS      1
2008    **      END **
```

```
The length of the code : 8


...Program finished with exit code 0
Press ENTER to exit console.
```

Symtbl.txt :



```
L1  2002
X   2007
```

```
The length of the code : 8


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion: We have designed suitable data structures and implemented pass-I of a two-pass assembler by including few instructions in the input and optab text file and few assembler directives.

Part A

SPOS Lab Assignment 2

Assignment No: 2	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Pass 2 of a two-pass assembler

Problem Statement: Implement pass-II of a two-pass assembler. The output of assignment-1 (intermediate file and symbol table) should be input for this assignment.

Theory: Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.

It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. If assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

- Pass 1
- Pass 2

Pass 2 Purpose: Generate object program

• Look up value of symbols(STGET)

• Generate instructions(MOTGET2)

• Generate data(for DS, DC and literals)

• Process psuedo ops(POTGET2)

Data structures: **Pass2:**

1. Copy of the source program input to pass 1.
2. Location counter(LC).
3. A table, MOT, that indicates for each instruction: symbol mnemonic, length, binary machine opcode, format.
4. A table, POT, that indicates for each psuedo op the symbol mnemonic and the action to be taken in pass 2.
5. The symbol table(ST), prepared by pass 1, containing each label and its corresponding value.
6. A table, the base table (BT), that indicates which registers are currently specified as base registers by USING psuedo-ops and the specified contents of these registers.
7. A workspace, INST, that is used to hold each instruction as its various parts are being assembled together.
8. A workspace, PRINT LINE, used to produce a printed listing.
9. A workspace, PUNCH CARD, used prior to actual outputting for converting the assembled instructions into the format needed by the loader.
10. An output program of assembled instructions in the format needed by the loader.

Format of Data structures :

- Pass 2 requires a MOT containing name, length, binary code and format; pass 1 requires only name and length
- Two separate tables can be used with different formats and contents or same table for both passes
- Same for POT
- Once we decide what information belongs in which database, format specification is necessary
- By format, we mean the format in which symbols are stored and the coding conventions

Program:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
```

```
public class Pass2 {
        public static void main(String[] Args) throws IOException{
                 BufferedReader b1 = new BufferedReader(new
FileReader("intermediate.txt"));
            BufferedReader b2 = new BufferedReader(new FileReader("symtab.txt"));
            BufferedReader b3 = new BufferedReader(new FileReader("littab.txt"));
            FileWriter f1 = new FileWriter("Pass2.txt");
            HashMap<Integer, String> symSymbol = new HashMap<Integer, String>();
            HashMap<Integer, String> litSymbol = new HashMap<Integer, String>();
            HashMap<Integer, String> litAddr = new HashMap<Integer, String>();
            String s;
            int symtabPointer=1,littabPointer=1,offset;
            while((s=b2.readLine())!=null){
                 String word[]=s.split("\t\t\t");
                 symSymbol.put(symtabPointer++,word[1]);
            }
            while((s=b3.readLine())!=null){
                 String word[]=s.split("\t\t");
                 litSymbol.put(littabPointer,word[0]);
                 litAddr.put(littabPointer++,word[1]);
            }
            while((s=b1.readLine())!=null){
                 if(s.substring(1,6).compareToIgnoreCase("IS,00")==0){
                         f1.write("+ 00 0 000\n");
                 }
                 else if(s.substring(1,3).compareToIgnoreCase("IS")==0){
                         f1.write("+ "+s.substring(4,6)+" ");
                         if(s.charAt(9)==')'){
                                 f1.write(s.charAt(8)+" ");
                                 offset=3;
                         }
                         else{
                                 f1.write("0 ");
                                 offset=0;
                         }
                         if(s.charAt(8+offset)=='S')

f1.write(symSymbol.get(Integer.parseInt(s.substring(10+offset,s.length()-1)))+"\n");
                         else

f1.write(litAddr.get(Integer.parseInt(s.substring(10+offset,s.length()-1)))+"\n");
                }
                 else if(s.substring(1,6).compareToIgnoreCase("DL,01")==0){
                         String s1=s.substring(10,s.length()-1),s2="";
                         for(int i=0;i<3-s1.length();i++)
                                 s2+="0";
                         s2+=s1;
                f1.write("+ 00 0 "+s2+"\n");
            }
            else{
                f1.write("\n");
            }
        }
        f1.close();
        b1.close();
        b2.close();
        b3.close();
    }
}
```

intermediate code -

```
(AD,01)(C,200)
(IS,04)(1)(L,1)
(IS,05)(1)(S,1)
(IS,04)(1)(S,1)
(IS,04)(3)(S,3)
(IS,01)(3)(L,2)
(IS,07)(6)(S,4)
(DL,01)(C,5)
(DL,01)(C,1)
(IS,02)(1)(L,3)
(IS,07)(1)(S,5)
(IS,00)
(AD,03)(S,2)+2
(IS,03)(3)(S,3)
(AD,03)(S,6)+1
(DL,02)(C,1)
(DL,02)(C,1)
(AD,02)
(DL,01)(C,1)
```

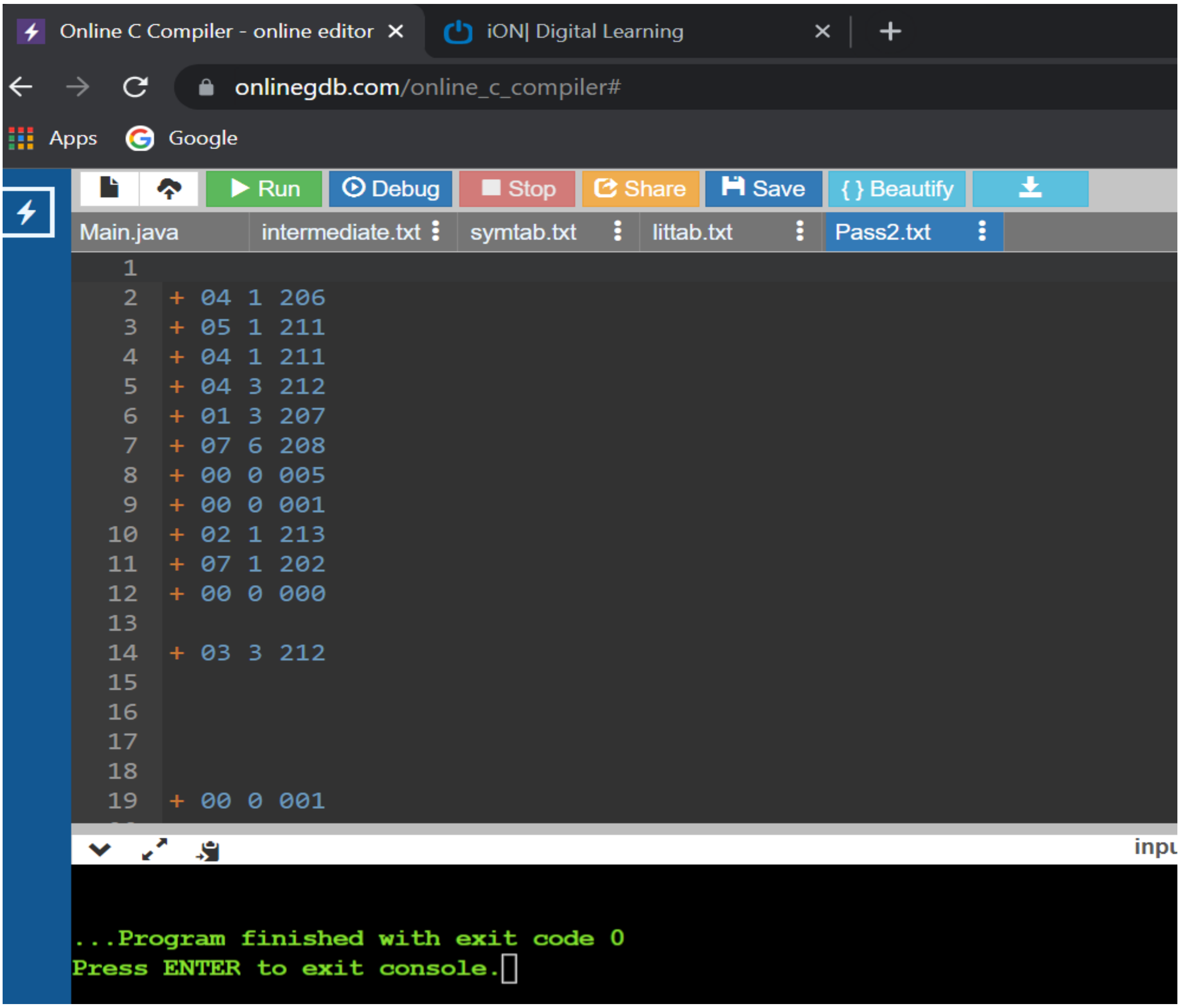
Symbol Table --

```
A               211             1
LOOP            202             1
B               212             1
NEXT            208             1
BACK            202             1
LAST            210             1
```

Literal Table --

```
5          206
1          207
1          213
```

Output: Machine code is the output



```

+ 04 1 206
+ 05 1 211
+ 04 1 211
+ 04 3 212
+ 01 3 207
+ 07 6 208
+ 00 0 005
+ 00 0 001
+ 02 1 213
+ 07 1 202
+ 00 0 000

+ 03 3 212




+ 00 0 001
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion : We have implement pass-II of a two-pass assembler. The output of assignment-1 (intermediate file, symbol table, literal table) were given as input for this program and we got machine code as the output.

Part A

SPOS Lab Assignment 3

Assignment No: 3	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Macro Program

Problem Statement: Design suitable data structures and implement macro definition and macro expansion processing for a sample macro with positional and keyword parameters.

Theory: A macro is a sequence of instructions, assigned by a name and could be used anywhere in the program.

Macro definition format :

Start of Definition-	MACRO
Macro name-	[ ]
Sequence to be abbreviated	- - - - - - - -
	- - - - - - - -
	- - - - - - - -
End of Definition-	MEND

Macro expansion: Expanding the macro call. This means that wherever we call a macro, at that point, we write the actual code present in the macro. For example in C, if we write #define SQUARE(X) X * X printf("%d", SQUARE(20)) then replacing the term "SQUARE(20)" with the actual definition of the macro i.e. "20 * 20" is called macro expansion.

Formal parameter: These are the variables declared in function definition, and receive their values when that function is called (Basically function parameters).

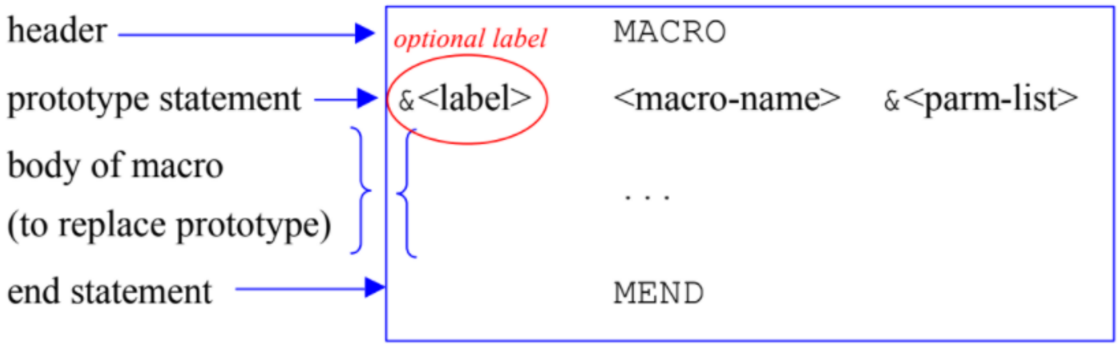
Actual parameter: The values/variables passed while calling a function are called actual parameters (Basically parameters when function is called).

In terms of C language, when we do #define SQUARE(X) X * X printf("%d", SQUARE(20)) Here, X is formal parameter (parameter passed in the function during definition), and 20 is the actual parameter (the argument of the macro when calling it)

Assembly language **macro definitions** can be predefined and placed in a macro library, or can be included "in-line" with the assembly language program.

The handling sequence for the program becomes:

Macro Definition Format



The prototype statement's parameter list as given to this point uses what are known as **positional parameters.** The position of the parameter in the comma separated list determines which entry it represents (note two successive commas in the list represents an omitted parameter).

A **keyword parameter** is one specified in the comma separated list by the form &<name>=<value> or &<name>= . In the first form, <value> is the default used in the macro's expansion if the programmer does not supply the parameter. In the second form, the programmer must supply the value (by name).

Program:

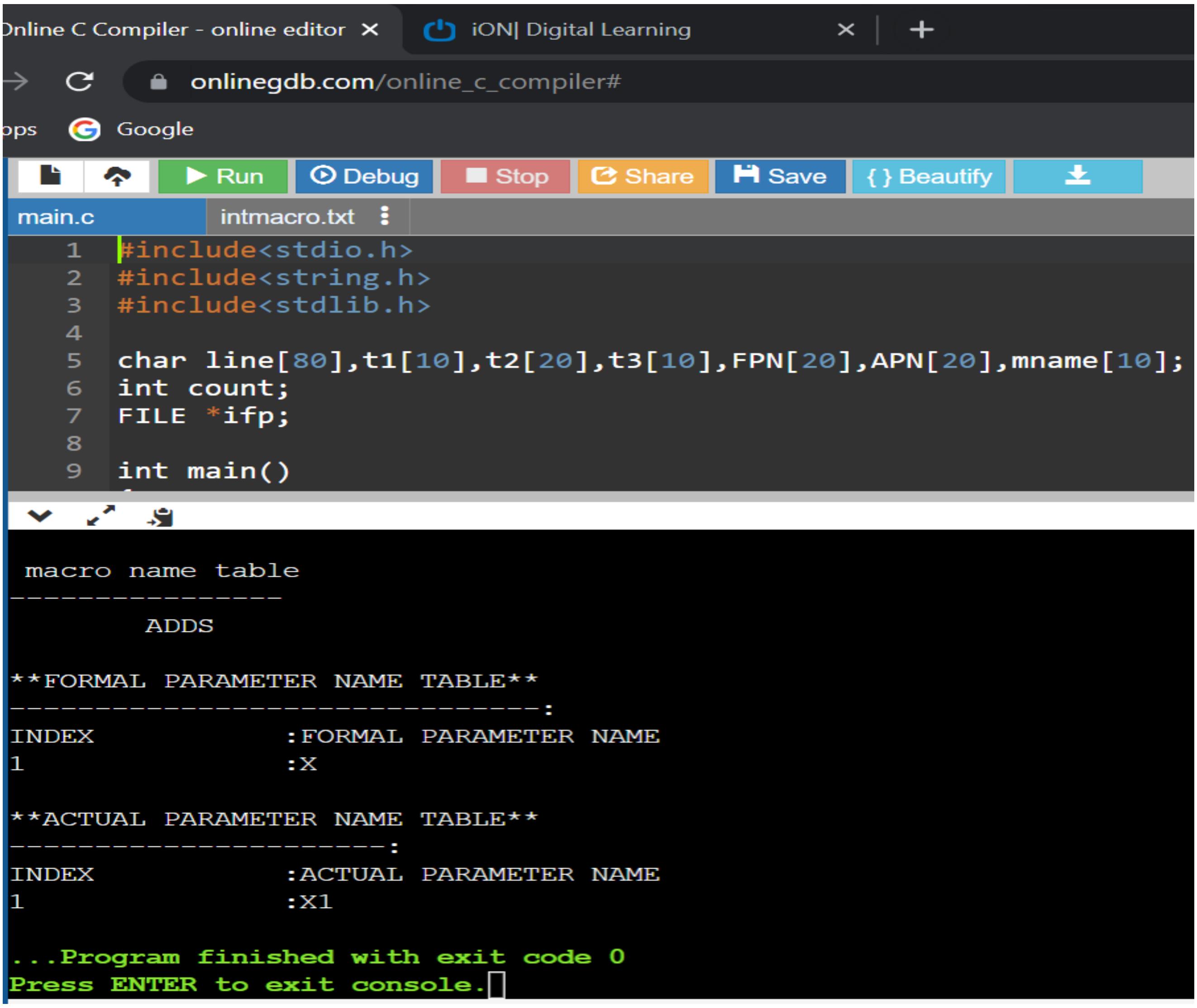
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
char line[80],t1[10],t2[20],t3[10],FPN[20],APN[20],mname[10];
int count;
FILE *ifp;
int main()
{
   int index=1;
   ifp= fopen("intmacro.txt","r");
   while(!feof(ifp))
   {
      fgets(line,20,ifp);
      count = sscanf(line,"%s%s%s",t1,t2,t3);
      if(strcmp("MACRO",t1)==0)
      {
         strcpy(mname,t2);
         printf("\n macro name table");
         printf("\n----------------\n");
         printf("\n%s",mname);
      }
      if(strcmp(mname,t2)==0)   {
         strcpy(FPN,t3);
         printf("\n\n\n**FORMAL PARAMETER NAME TABLE**");
         printf("\n-------------------------------:\n");
         printf("\nINDEX\t\t:FORMAL PARAMETER NAME");
         printf("\n%d\t:%s",index,FPN); }
      if(strcmp(mname,t1)==0) {
         strcpy(APN,t2);
         printf("\n\n\n**ACTUAL PARAMETER NAME TABLE**");
         printf("\n----------------------:\n");
         printf("\nINDEX\t\t:ACTUAL PARAMETER NAME");
         printf("\n%d\t:%s",index,APN);
```

}}}

Intmacro:

MACRO ADDS X
ADD AREG BREG MEND
START 100
MOVER AREG CREG
ADDS X1
SUB AREG CREG
END

Output:



```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

char line[80],t1[10],t2[20],t3[10],FPN[20],APN[20],mname[10];
int count;
FILE *ifp;

int main()
```

```
 macro name table
----------------
        ADDS

**FORMAL PARAMETER NAME TABLE**
-------------------------------:
INDEX               :FORMAL PARAMETER NAME
1                   :X

**ACTUAL PARAMETER NAME TABLE**
----------------------:
INDEX               :ACTUAL PARAMETER NAME
1                   :X1

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion : We have designed suitable data structures and implemented macro definition and macro expansion processing for a sample macro with positional and keyword parameters.

Part B

SPOS Lab Assignment 4

Assignment No: 4	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Menu Driven program using shell scripting

Problem Statement: Implement the following using shell scripting Menu driven program for :

a. Find the factorial of a no.
b. Find greatest of three numbers
c. Find a prime no
d. Find whether a number is palindrome
e. Find whether a string is palindrome

Theory:A menu-driven shell script provides users more options/interactive interface.

In a layman's term shell script is similar to the restaurant menu, suppose you are at your favorite restaurant, and you asked for a restaurant menu, so you can choose your favorite dish.Similarly, a menu-driven shell script serves the same purpose.

Using case esac statement to create a menu-driven shell script:

- A case statement is similar to a switch statement from another languages.
- Case statement is an alternative to multilevel if-then-else statements.
- Using case statement we can avoid using multiple if-then-else and reduce script size.
- Case is used to match several value against one value.

Syntax:

```
case $condition in
pattern1          Statement(s) to be executed if pattern1 matches;;
pattern2          Statement(s) to be executed if pattern2 matches;;
*)  Default case;;
esac
```

- Here, case is keyword and $condition is the input value that we match again several another values.
- Pattern can be anything, it can be alphabets, integers or regexp. pattern is terminated with the closing parenthesis ‘)‘.
- If the pattern match with the input value then statements followed by the pattern will be executed and the statements are terminated by two semicolons ‘;;‘.
- *) is default case i.e. if user input does not match any of the given pattern the statements followed by this pattern will be executed.
- To end the case statement esac keyword is used. It is the reverse of ‘case’ keyword.

Steps:

1. Create a custom menu using **echo** statement and show the menu
2. Create an infinite loop using **while** statement that accept the user input option and generate the output continuously until the user input matches the exit pattern.
3. Take input from the user using **read** statement and store it in a variable.
4. Use case statement to check if the input matches with the pattern.
5. Create custom pattern.
6. Exit the case statement using **esac** keyword.

Program:

```
fact()  {

echo "Enter a number"

read num

fact=1

while [ $num -gt 1 ]

do

fact=$((fact * num))      #fact = fact * num

 num=$((num - 1))        #num = num - 1
```

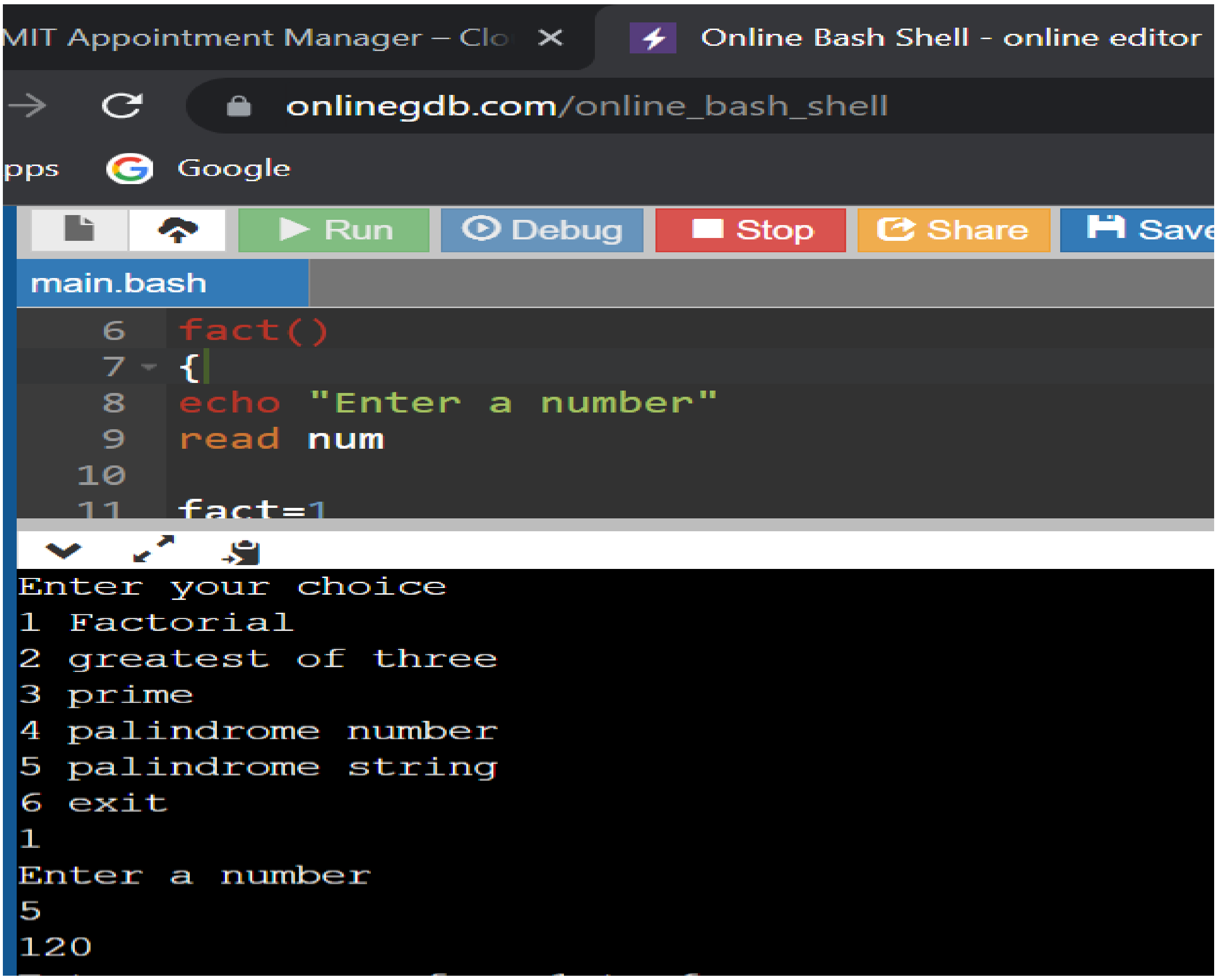
```
done
echo $fact
}
greatest_of_three()
{
echo "Enter num1"
read num1
echo "Enter num2"
read num2
echo "Enter num3"
read num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
    echo "$num1 is the greatest"
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
    echo "$num2 is the greatest"
else
    echo "$num3 is the greatest"
fi
}
prime()
{
echo "Enter a number"
read num
for((i=2; i<=num/2; i++))
do
  if [ $((num%i)) -eq 0 ]
  then
    echo "$num is not a prime number."
    exit
  fi
done
echo "$num is a prime number."
}
palindrome_num()
{
echo "Enter a Number"
read num
rev=""
   temp=$num
while [ $num -gt 0 ]
do
    s=$(( $num % 10 ))
    num=$(( $num / 10 ))
    rev=$( echo ${rev}${s} )
done
if [ $temp -eq $rev ];
then
    echo "$temp is palindrome"
else
    echo "$temp is NOT palindrome"
fi
}
palindrome_string()
{
```

```
echo "Input the string without space"
read str
rev=$(echo $str | rev)
echo"The given string is " $str
echo"Its reverse is "$rev
if [ "$str" = "$rev" ]; then
   echo "It is a palindrome."
else
   echo "It is not a palindrome."
fi
}
while :
do
echo "Enter your choice"
echo "1 Factorial"
echo "2 greatest of three"
echo "3 prime"
echo "4 palindrome number"
echo "5 palindrome string"
echo "6 exit"
read ch
case $ch in
1) fact;;
2) greatest_of_three;;
3) prime;;
4) palindrome_num;;
5) palindrome_string;;
6) exit;;
```

```
*) echo  "invalid option"
esac
echo "Enter your menu from 1 to 6"
done
```

<u>Output:</u>

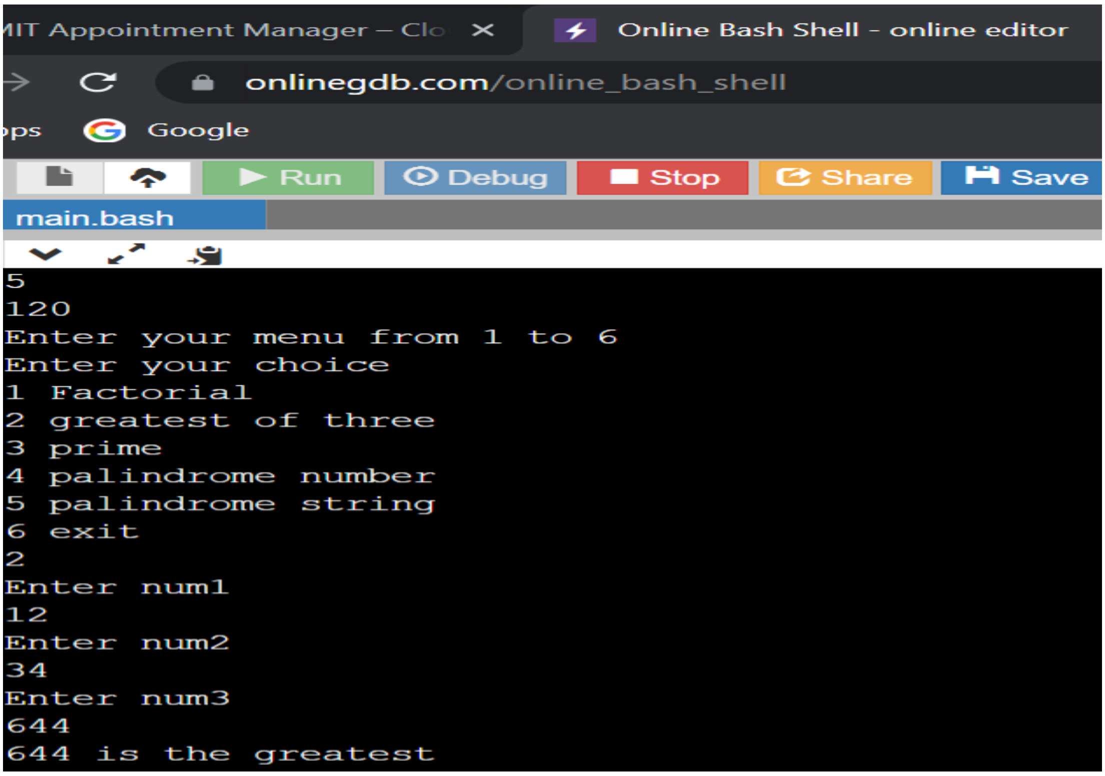
Factorial:


MIT Appointment Manager – Clo
Online Bash Shell - online editor
onlinegdb.com/online_bash_shell
pps
Google
Run
Debug
Stop
Share
Save
main.bash

```
fact()
{
echo "Enter a number"
read num

fact=1
```

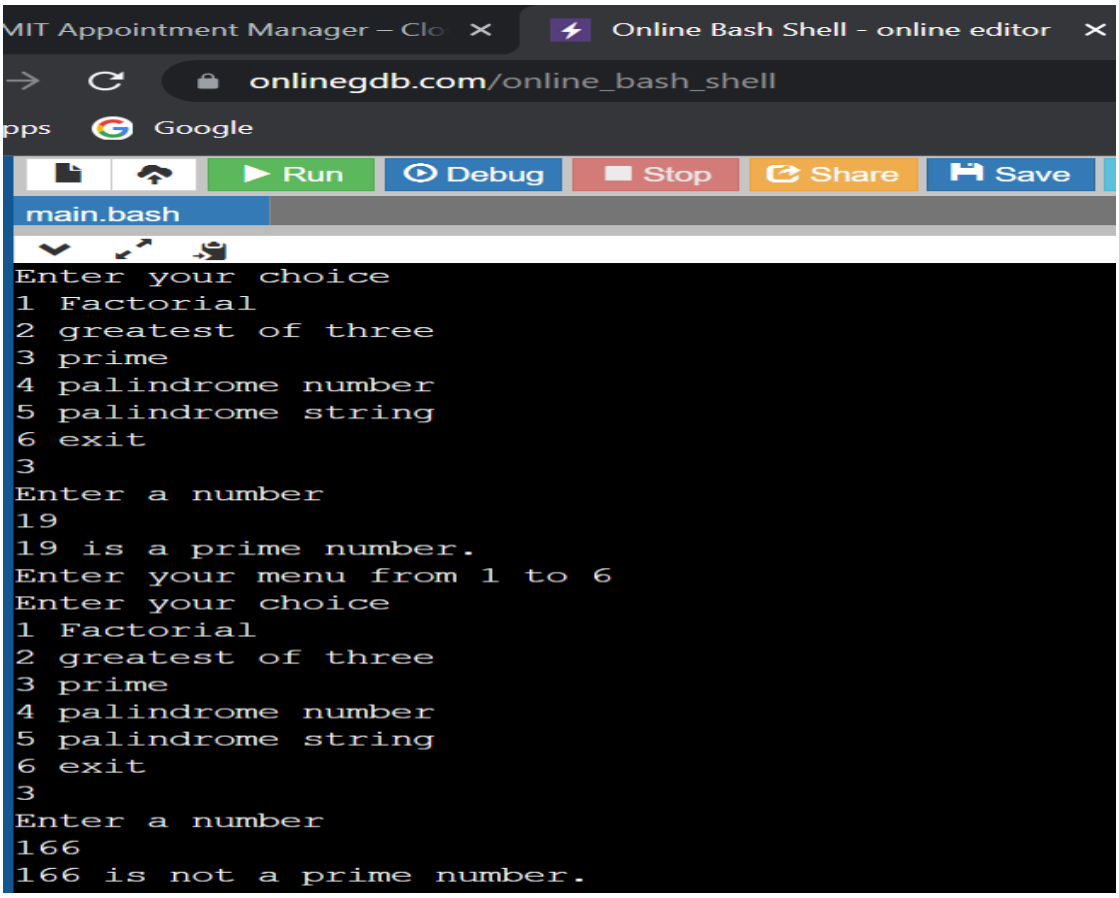
```
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
1
Enter a number
5
120
```

Greatest of three:


MIT Appointment Manager – Clo
Online Bash Shell - online editor
onlinegdb.com/online_bash_shell
pps
Google
Run
Debug
Stop
Share
Save
main.bash

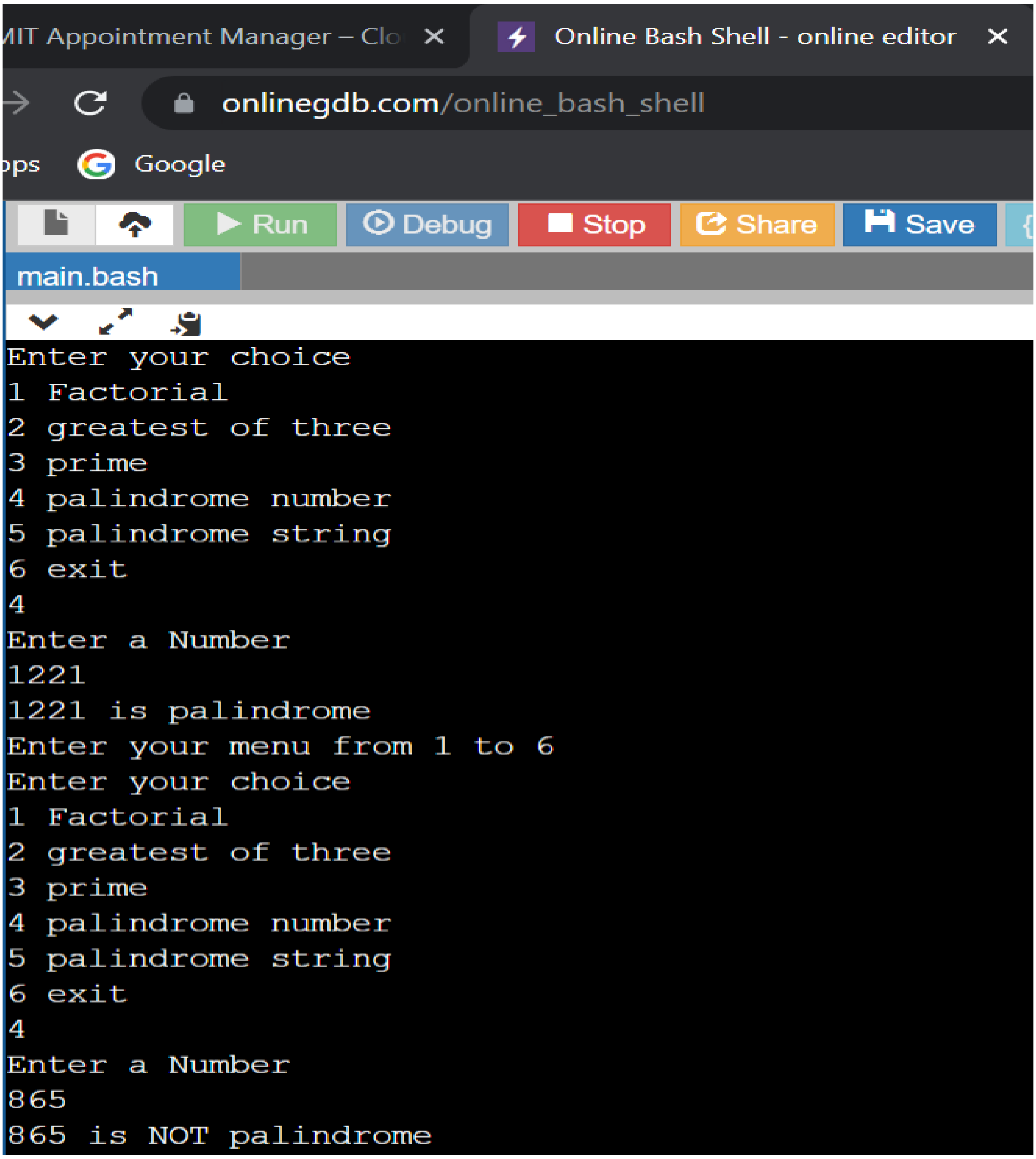
```
5
120
Enter your menu from 1 to 6
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
2
Enter num1
12
Enter num2
34
Enter num3
644
644 is the greatest
```

Prime number:


MIT Appointment Manager – Clo
Online Bash Shell - online editor
onlinegdb.com/online_bash_shell
pps
Google
Run
Debug
Stop
Share
Save
main.bash

```
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
3
Enter a number
19
19 is a prime number.
Enter your menu from 1 to 6
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
3
Enter a number
166
166 is not a prime number.
```

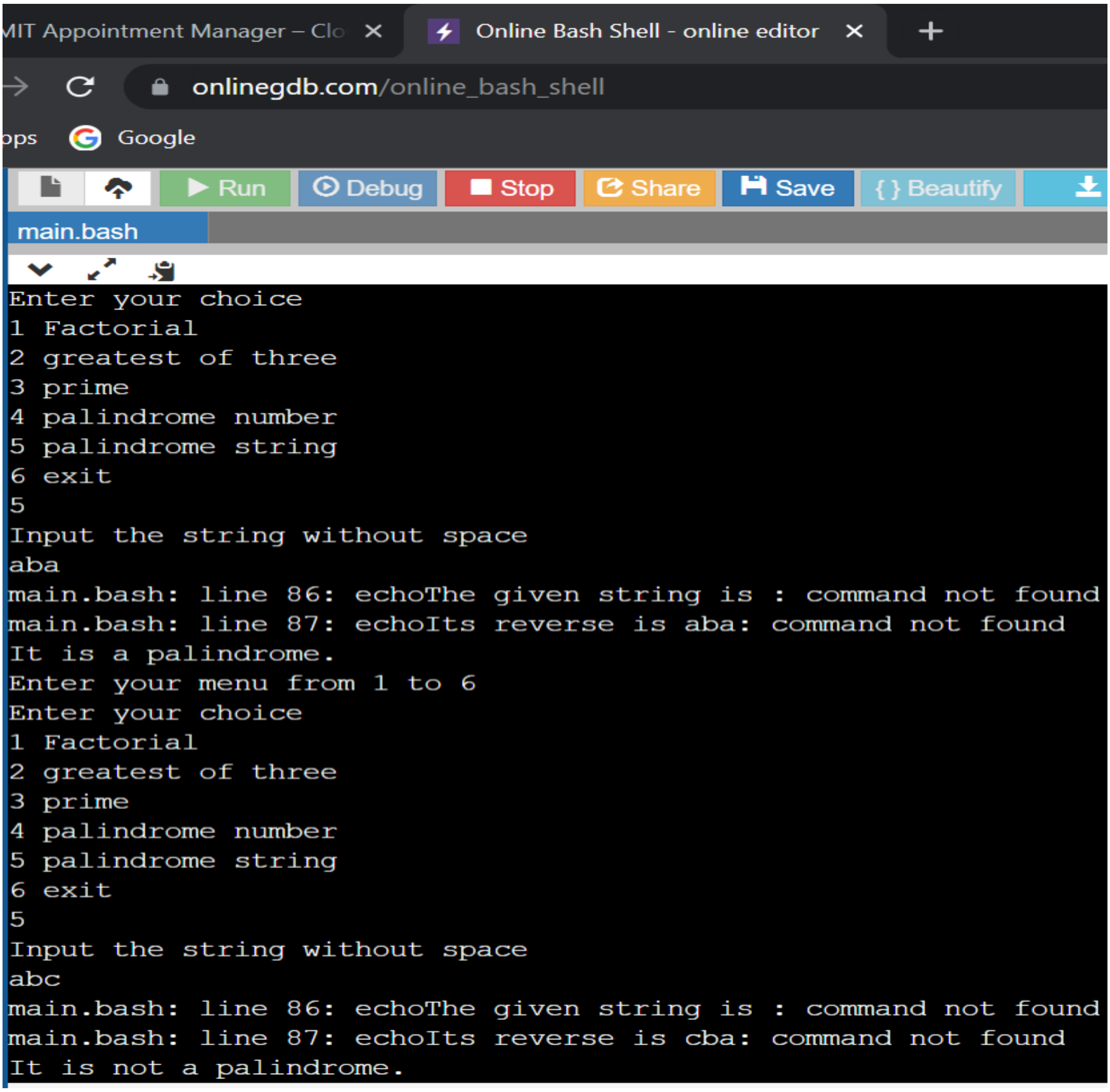
Palindrome String:



MIT Appointment Manager – Clo
Online Bash Shell - online editor
onlinegdb.com/online_bash_shell
ps
Google
Run
Debug
Stop
Share
Save
main.bash

```
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
4
Enter a Number
1221
1221 is palindrome
Enter your menu from 1 to 6
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
4
Enter a Number
865
865 is NOT palindrome
```

Palindrome Number:


MIT Appointment Manager – Clo
Online Bash Shell - online editor
onlinegdb.com/online_bash_shell
ops
Google
Run
Debug
Stop
Share
Save
{ } Beautify
main.bash

```
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
5
Input the string without space
aba
main.bash: line 86: echoThe given string is : command not found
main.bash: line 87: echoIts reverse is aba: command not found
It is a palindrome.
Enter your menu from 1 to 6
Enter your choice
1 Factorial
2 greatest of three
3 prime
4 palindrome number
5 palindrome string
6 exit
5
Input the string without space
abc
main.bash: line 86: echoThe given string is : command not found
main.bash: line 87: echoIts reverse is cba: command not found
It is not a palindrome.
```

Conclusion: We have implemented the following using shell scripting Menu driven program for : factorial of a no, greatest of three numbers, prime no, whether a number is palindrome or not, whether a string is palindrome or not.

Part B

SPOS Lab Assignment 5

Assignment No: 5	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Process control system calls.

Problem Statement: Process control system calls - Fork, execve and wait system calls along with the demonstration of zombie and orphan states.

1. Application should consist of Fork –wait combination (parent with one application and child with another application) and students must demonstrate zombie and orphan states.
2. Application should consist of Fork –execve combination (parent with one application and child with another application).

Theory:

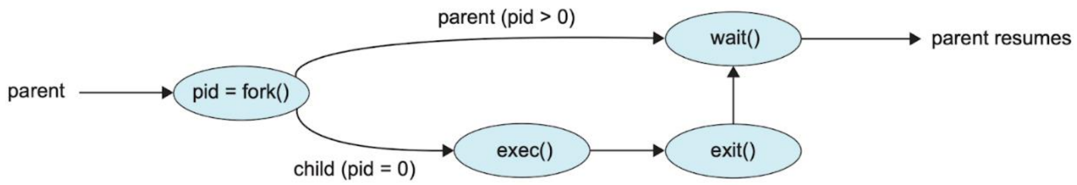


Figure 3.10 Process creation using the `fork()` system call.

In most cases, we may want to execute a different program in the child process than the parent. The exec() family of function calls creates a new address space and loads a new program into it.
Finally, a process exits or terminates using the exit() system call. This function frees all the resources held by the process(except for pcb). A parent process can enquire about the status of a terminated child using wait() system call. When the parent process uses wait() system call, the parent process is blocked till the child on which it is waiting terminates.

Zombie Process
Zombie process is that process which has terminated but whose process control block has not been cleaned up from main memory because the parent process was not waiting for the child. If there are a large number of zombie processes, their PCBs IN WORST CASE can occupy the whole RAM

Orphan Process
Orphan process is a process that doesn’t have a parent process. This can happen when the parent process terminates due to some reasons before the completion of the child**.**

System call **fork()** is used to create processes. It takes no arguments and returns a process ID. The purpose of **fork()** is to create a ***new*** process, which becomes the *child* process of the caller. After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()*** system call. Therefore, we have to distinguish the parent from the child.
This can be done by testing the returned value of **fork()**:

- If **fork()** returns a negative value, the creation of a child process was unsuccessful.
- **fork()** returns a zero to the newly created child process.
- **fork()** returns a positive value, the ***process ID*** of the child process, to the parent. The returned process ID is of type **pid_t** defined in **sys/types.h**. Normally, the process ID is an integer. Moreover, a process can use function **getpid()** to retrieve the process ID assigned to this process.

Fork() System Call:
Therefore, after the system call to **fork()**, a simple test can tell which process is the child. **Please note that Unix will make an exact copy of the parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces**.
Therefore, after the system call to fork(), a simple test can tell which process is the child. Note that Unix will make an exact copy of the parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces.

Program: *Program 1:*

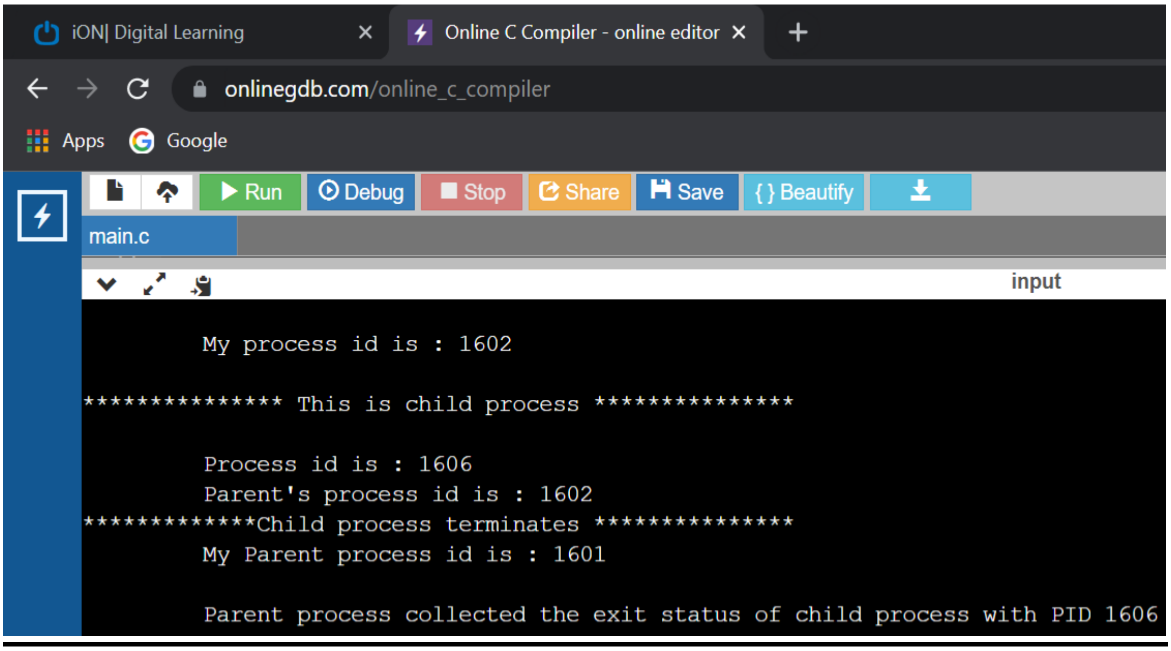
```
#include <stdio.h>

#include<sys/types.h>  //fork, sleep, getpid, getppid
```

```
#include<sys/wait.h>    //system call - wait
#include<stdlib.h>
#include<unistd.h>
int main()
{
        pid_t cpid;    //Declaring a variable cpid with the pid_t  data type
        int  *status=NULL;        //Intilizating a pointer var status to NULL
cpid = fork()  //The process fork_wait creates a new process --clone
if( cpid == 0 )
{    //CHILD PROCESS as it is not creating any new process
  printf("\n*************** This is child process ***************\n ");
//write sys call
                printf("\n\t Process id is : %d", getpid());
                printf("\n\t Parent's process id is : %d", getppid());
                sleep(15);
        printf("\n*************Child process terminates ***************\n");
        }
        else { /*Parent process waiting for child process, to complete the
task*/
          printf("\n\t My process id is : %d", getpid());
          printf("\n\t My Parent process id is : %d", getppid());
cpid = wait(status);   //Forceful wait; that collects the exit status of child
process with cpid
printf("\n\n\t Parent process collected the exit status of child process with
PID %d\n\n", cpid);
}    //end of if-else
return 0;
}    //end of main
```

Program 2:

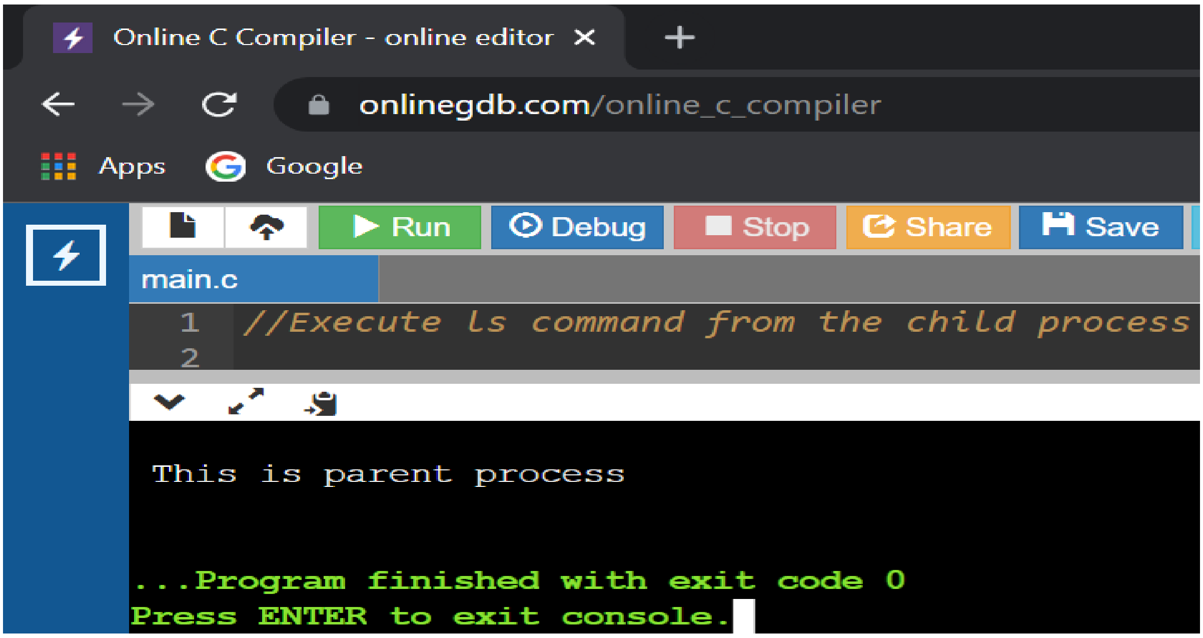
```
//Execute ls command from the child process

#include <unistd.h>       //for execv

#include <sys/types.h>

#include <stdio.h>

#include <stdlib.h>

int main(){

        pid_t pid;

        int arr[10];

char * argv_list[] = {"ls","-lrt","/home/hp/Botnet", NULL}; // ls -lart /home/
NULL - no more arg

      pid=fork();

      if(pid==0)  //child process

        {

                execv("/bin/ls",argv_list); //arg1 - path of the executable
                printf("\n This is child \n");

        }

        else  //parent process

        {

                //sleep(30);

                printf("\n This is parent process\n");

        }

}
```

Output: Program 1:



```
         My process id is : 1602

*************** This is child process ***************

         Process id is : 1606
         Parent's process id is : 1602
*************Child process terminates ***************
         My Parent process id is : 1601

         Parent process collected the exit status of child process with PID 1606
```

Program 2:


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Share
Save
main.c

```c
//Execute ls command from the child process

```

```
 This is parent process


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion: We have implemented Process control system calls - Fork, execve and wait system calls along with the demonstration of zombie and orphan states.

Part B

SPOS Lab Assignment 6

Assignment No: 6	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

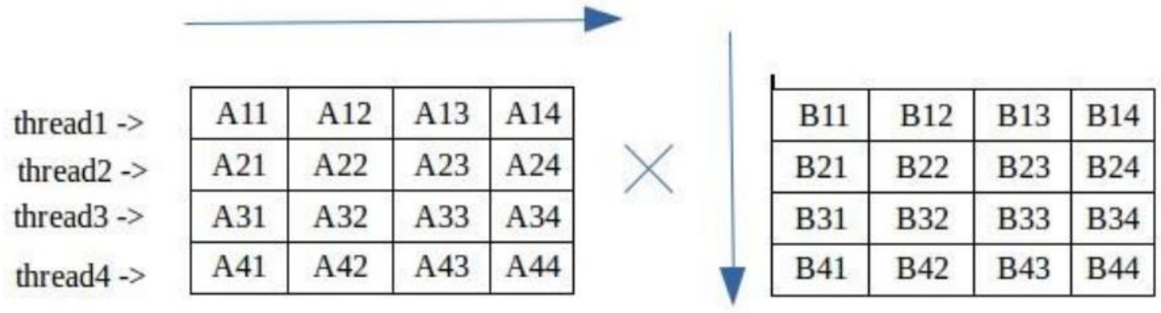
Title: Matrix multiplication using multithreading

Problem Statement: Implement matrix multiplication using multithreading with pthread library.

Theory: Multiplication of matrix does take time surely. Time complexity of matrix multiplication is O(n^3) using normal matrix multiplication. And Strassen algorithm improves it and its time complexity is O(n^(2.8074)). But, Is there any way to improve the performance of matrix multiplication using the normal method.

Multi-threading can be done to improve it. In multi-threading, instead of utilizing a single core of your processor, we utilizes all or more core to solve the problem. We create different threads, each thread evaluating some part of matrix multiplication. Depending upon the number of cores your processor has, you can create the number of threads required. In second approach,we create a separate thread for each element in resultant matrix.

Using pthread_exit() we return computed value from each thread which is collected by pthread_join(). This approach does not make use of any global variables.



thread1 ->	A11	A12	A13	A14
thread2 ->	A21	A22	A23	A24
thread3 ->	A31	A32	A33	A34
thread4 ->	A41	A42	A43	A44

B11	B12	B13	B14
B21	B22	B23	B24
B31	B32	B33	B34
B41	B42	B43	B44

A **thread** is defined as an independent stream of instructions that can be scheduled to run as such by the operating system.A Thread Group is a set of threads all executing inside the same process. They all share the same memory, and thus can access the same global variables, same heap memory, same set of file descriptors, etc. All these threads execute in parallel (i.e. using time slices).

Multithreading is a CPU feature that allows two or more instruction threads to execute independently while sharing the same process resources.

The POSIX thread libraries are a standards based thread API for C/C++. Full form of POSIX threads is Portable Operating System Interface.It is highly concrete multithreading system and it is the default standard of UNIX system. POSIX is a kind of interface that Operating system needs to support.

The **pthread_create function** to create a real, living thread.

The call to **pthread_exit** causes the current thread to exit and free any thread-specific resources it is taking.

pthread_create - thread creation

#include <pthread.h>
int pthread_create(pthread_t **thread*, const pthread_attr_t **attr*,
void *(**start_routine*)(void*), void **arg*);

- The *pthread_create()* function is used to create a new thread, with attributes specified by *attr*, within a process.
- If *attr* is NULL, the default attributes are used. If the attributes specified by *attr* are modified later, the thread's attributes are not affected.

- Upon successful completion, *pthread_create()* stores the ID of the created thread in the location referenced by *thread*.
- The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine* returns, the effect is as if there was an implicit call to *pthread_exit()* using the return value of *start_routine* as the exit status.
- Note that the thread in which *main()* was originally invoked differs from this. When it returns from *main()*, the effect is as if there was an implicit call to *exit()* using the return value of *main()* as the exit status.
- The signal state of the new thread is initialised as follows:
  - ➢ The signal mask is inherited from the creating thread.
  - ➢ The set of signals pending for the new thread is empty.
- If *pthread_create()* fails, no new thread is created and the contents of the location referenced by *thread* are undefined.

pthread_join - wait for thread termination

#include <pthread.h> int pthread_join(pthread_t *thread*, void ****value_ptr*);

DESCRIPTION

- The pthread_join() function suspends execution of the calling thread until the target thread terminates, unless the target thread has already terminated.
- On return from a successful pthread_join() call with a non-NULL value_ptr argument, the value passed to pthread_exit() by the terminating thread is made available in the location referenced by value_ptr.
- When a pthread_join() returns successfully, the target thread has been terminated.
- The results of multiple simultaneous calls to pthread_join() specifying the same target thread are undefined.
- If the thread calling pthread_join() is canceled, then the target thread will not be detached.

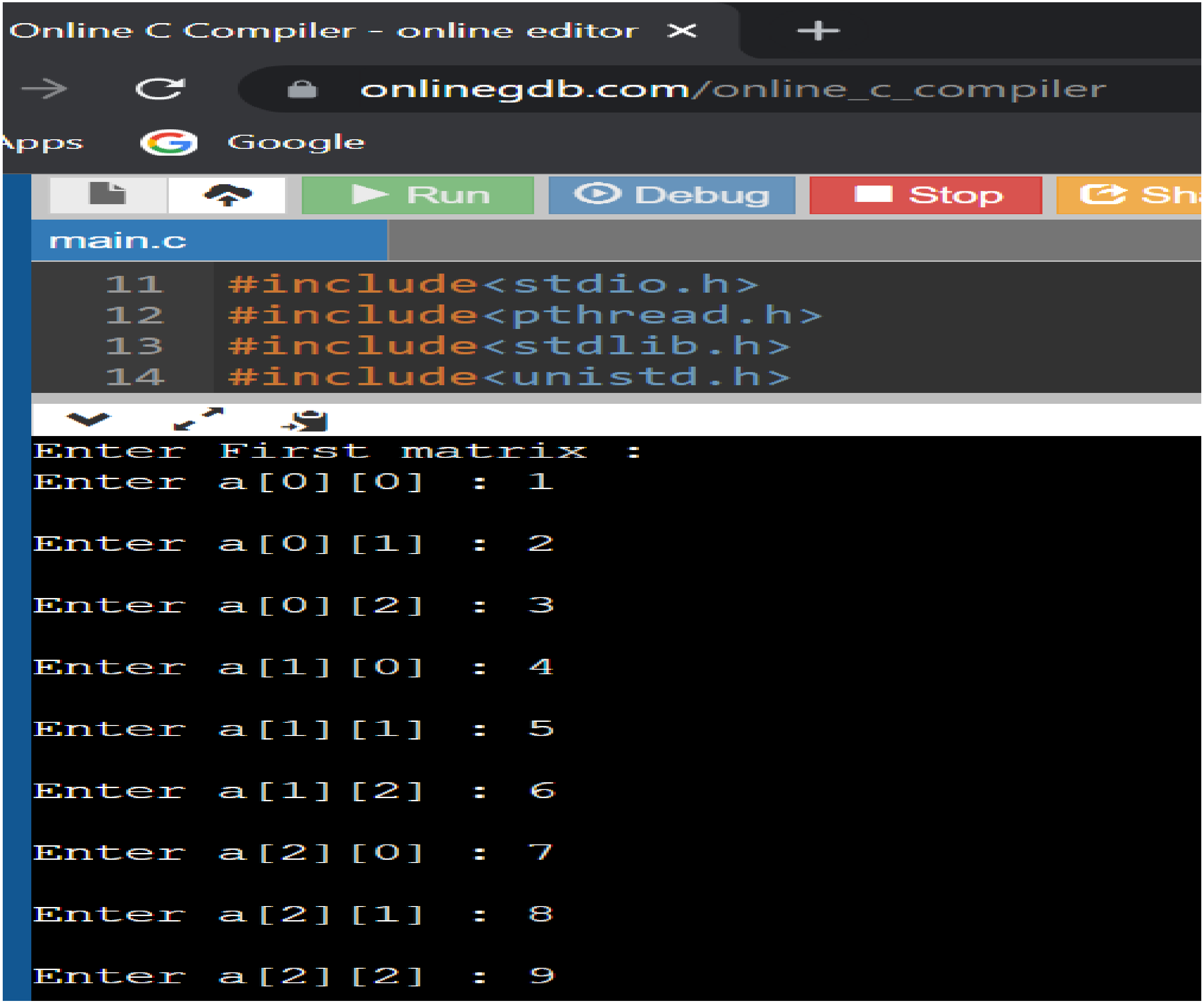
It is unspecified whether a thread that has exited but remains unjoined counts against _POSIX_THREAD_THREADS_MAX.

RETURN VALUE
If successful, the pthread_join() function returns zero. Otherwise, an error number is returned to indicate the error.

Program: Implement matrix multiplication using multithreading with pthread library.

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#define MAX 3

int a[MAX][MAX];
int b[MAX][MAX];
int c[MAX][MAX];

//Generic function prototypes
void* mult(void*);
int main()
{
   pthread_t tid1,tid2,tid3; //Threads for row1, row2, and row3
   int row1,row2,row3;
   int i,j;
printf("\n\nEnter First matrix : ");
   for(i=0;i<MAX;i++){
      for(j=0;j<MAX;j++){
         printf("\nEnter a[%d][%d] : ",i,j);
         scanf("%d",&a[i][j]);
      }
   }
printf("\n\nEnter Second matrix");
   for(i=0;i<MAX;i++){
        for(j=0;j<MAX;j++){
            printf("\nEnter b[%d][%d] : ",i,j);
            scanf("%d",&b[i][j]);
        }
    }
row1=0;
//Create a thread as tid1; executing mult routine; accepting index of row1
   pthread_create(&tid1, NULL, mult, &row1);  // pthread_create(a1, a2,
mult, 0)
row2=1;
//Create a thread as tid2; executing mult routine; accepting index of row2
   pthread_create(&tid2, NULL, mult, &row2);
row3=2;
//Create a thread as tid3; executing mult routine; accepting index of row3
    pthread_create(&tid3, NULL, mult, &row3);
             pthread_join(tid1,NULL);
             pthread_join(tid2,NULL);
             pthread_join(tid3,NULL);
 printf("\n\nResult is : \n");
    for(i=0;i<MAX;i++){
       for(j=0;j<MAX;j++){
          printf("%d ",c[i][j]);
       }
       printf("\n");
    }
 exit(0); }  //End of main
void* mult(void * arg)  // Address of 0th row
{
        int i,j,k;
        i = *(int * )arg;  //value of i = 0, 1, or 2
        for(j=0;j<MAX;j++){
                c[i][j] = 0;
                for(k=0;k<MAX;k++){
                        c[i][j] += a[i][k] * b[k][j];
                }
        }
        printf("\nThread id is : %ld",pthread_self());
}//END
```

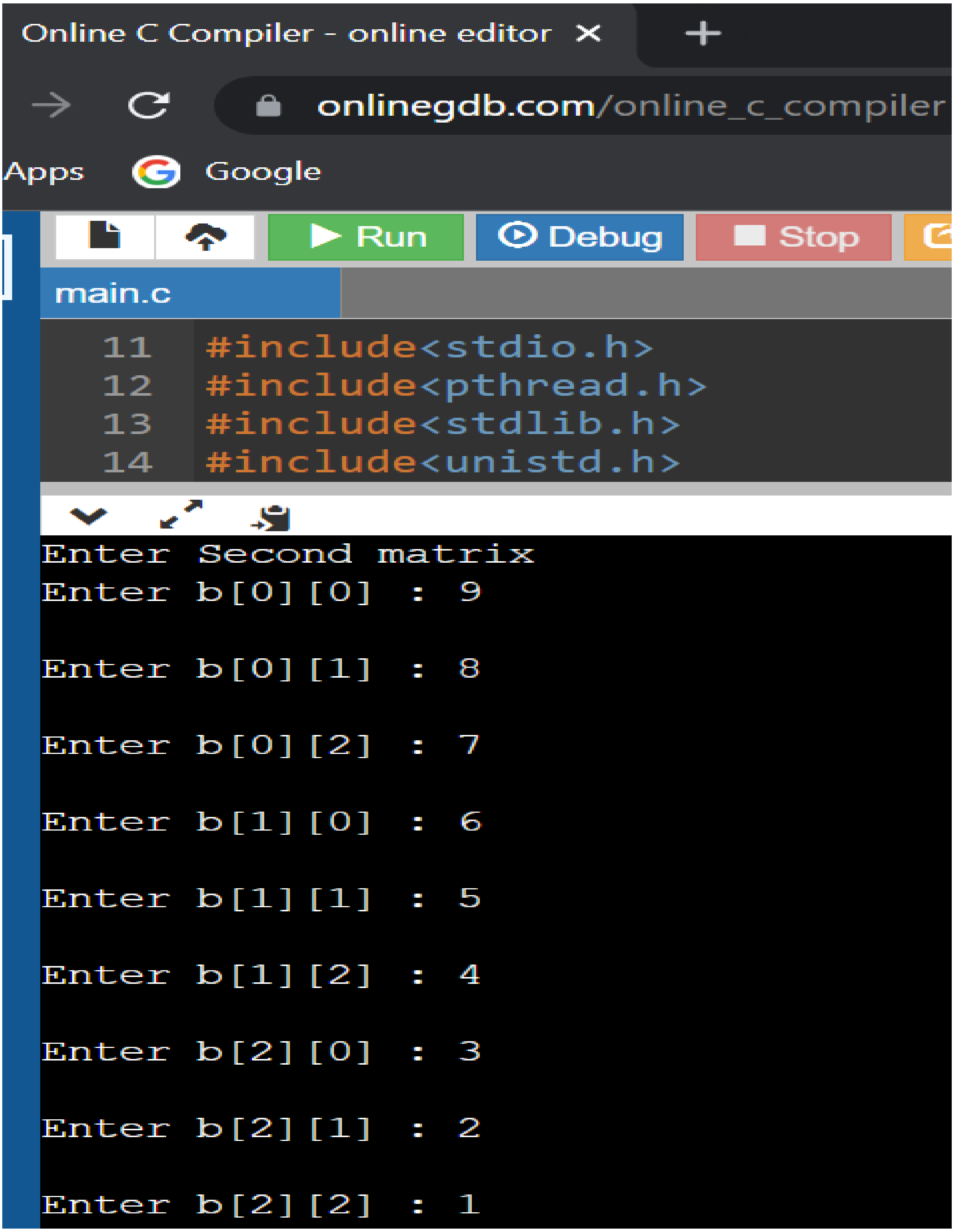
<u>Output:</u> Matrix 1 input:


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Sh
main.c

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
```

```
Enter First matrix :
Enter a[0][0] : 1

Enter a[0][1] : 2

Enter a[0][2] : 3

Enter a[1][0] : 4

Enter a[1][1] : 5

Enter a[1][2] : 6

Enter a[2][0] : 7

Enter a[2][1] : 8

Enter a[2][2] : 9
```

Matrix 2 input:

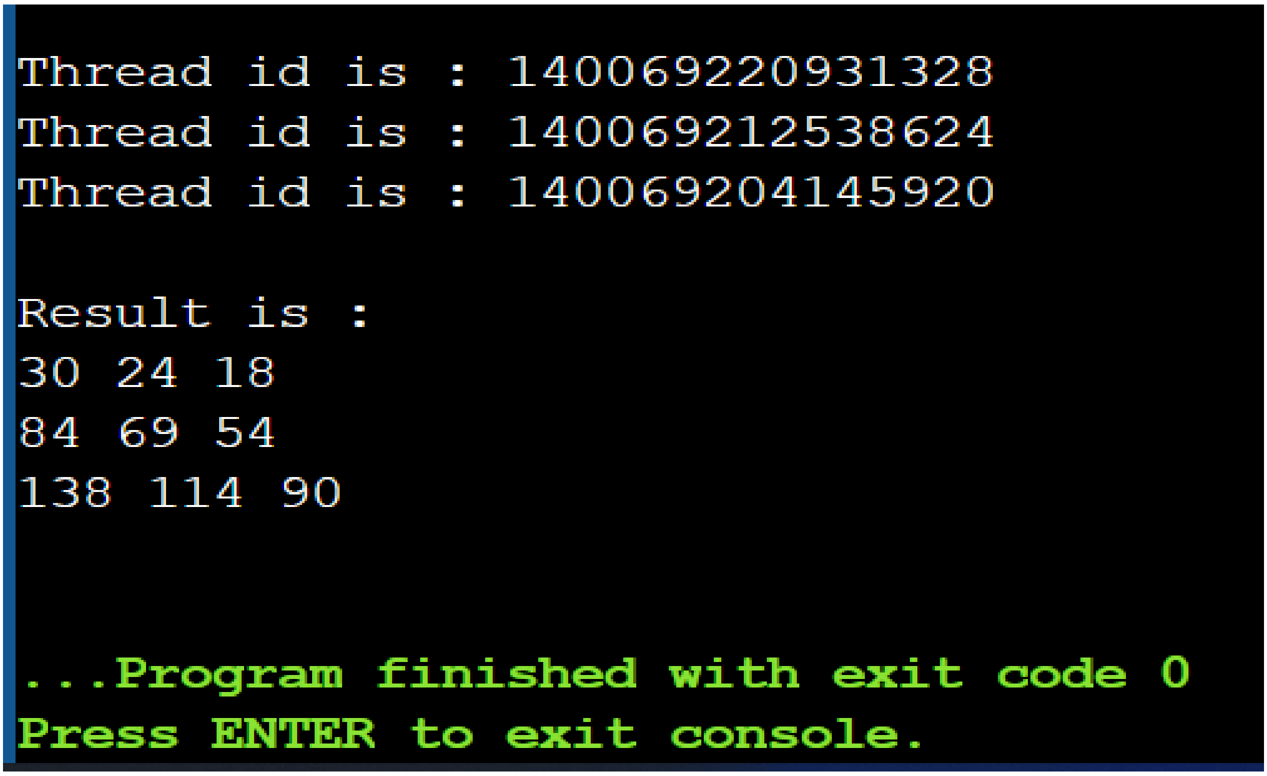


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
main.c

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
```

```
Enter Second matrix
Enter b[0][0] : 9

Enter b[0][1] : 8

Enter b[0][2] : 7

Enter b[1][0] : 6

Enter b[1][1] : 5

Enter b[1][2] : 4

Enter b[2][0] : 3

Enter b[2][1] : 2

Enter b[2][2] : 1
```

Result of matrix multiplication:



```
Thread id is : 140069220931328
Thread id is : 140069212538624
Thread id is : 140069204145920

Result is :
30 24 18
84 69 54
138 114 90


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion: We have implemented matrix multiplication with multithreading using pthread library.

Part B

SPOS Lab Assignment 7

Assignment No: 7	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Scheduling Algorithms

Problem Statement: Simulate the following scheduling algorithms using 'C'/Python/Java language.

a) FCFS (Non-preemptive by default)
b) SRTF (Preemptive version of SJF)
c) Priority (Non-preemptive)
d) Round Robin (Preemptive by default)

Theory:

What is CPU Scheduling Algorithm?

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Types of CPU Scheduling:

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

Data structures:

- ***CPU Utilization*****:** To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

- ***Throughput:*** It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

- ***Turnaround Time*****:** It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process (Wall clock time).

- ***Waiting Time:*** The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

- ***Response Time***: Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

FCFS: FCFS stands for First Come First Serve. In the FCFS scheduling algorithm, the job that arrived first in the ready queue is allocated to the CPU and then the job that came second and so on.

SRTF: SRTF, Which Stands for Shortest Remaining Time First is a scheduling algorithm used in Operating Systems, which can also be called as the preemptive version of the SJF scheduling algorithm. The process which has the least processing time remaining is executed first.

Priority (Non-preemptive) : Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned first arrival time (less arrival time process first) if two processes have same arrival time, then compare to priorities (highest process first). Also, if two processes have same priority then compare to process number (less process number first). This process is repeated while all process get executed.

Round Robin (Preemptive by default): Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
- One of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.

Program:

❖ FCFS (Non-preemptive by default) using C language:

```
// C program for implementation of FCFS scheduling
#include<stdio.h>
// Function to find the waiting time for all processes
void findWaitingTime(int processes[], int n,  int bt[], int wt[])
{
 // waiting time for first process is 0
    wt[0] = 0;
   // calculating waiting time
    for (int  i = 1; i < n ; i++ )
        wt[i] =  bt[i-1] + wt[i-1] ;
}
// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n,int bt[], int wt[], int tat[])
{
 // calculating turnaround time by adding   bt[i] + wt[i]
    for (int  i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}
//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
  //Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt);
 //Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);
//Display processes along with all details
    printf("Processes   Burst time   Waiting time   Turn around time\n");
//Calculate total waiting time and total turn around time
    for (int  i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("   %d ",(i+1));
        printf("       %d ", bt[i] );
        printf("       %d",wt[i] );
        printf("       %d\n",tat[i] );
    }
    int s=(float)total_wt / (float)n;
    int t=(float)total_tat / (float)n;
    printf("Average waiting time = %d",s);
    printf("\n");
    printf("Average turn around time = %d ",t);
}
// Driver code
```

```
int main()
{
  //process id's
    int processes[] = { 1, 2, 3};
    int n = 3;
  //Burst time of all processes
    int  burst_time[] = {10, 5, 8};
    findavgTime(processes, n,  burst_time);
    return 0;
}
```

❖ SRTF (Preemptive version of SJF) using C++:

```
// C++ program to implement Shortest Remaining Time First
  #include <bits/stdc++.h>
   using namespace std;
 struct Process {
    int pid; // Process ID
    int bt; // Burst Time
    int art; // Arrival Time
};
// Function to find the waiting time for all processes
void findWaitingTime(Process proc[], int n, int wt[])
{
    int rt[n];
// Copy the burst time into rt[]
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;
    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;
// Process until all processes gets completed
    while (complete != n) {
    //Find process with minimum remaining time among the processes
that arrives till the current time
        for (int j = 0; j < n; j++) {
            if ((proc[j].art <= t) &&
            (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }
         if (check == false) {
            t++;
            continue;    }
// Reduce remaining time by one
        rt[shortest]--;
 // Update minimum
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;
// If a process gets completely executed
        if (rt[shortest] == 0) {
// Increment complete
            complete++;
            check = false;
 // Find finish time of current process
            finish_time = t + 1;
// Calculate waiting time
            wt[shortest] = finish_time -
                        proc[shortest].bt -
                        proc[shortest].art;
           if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
// Increment time
        t++;
    }
}
// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n,int wt[], int tat[])
{
 // calculating turnaround time by adding bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
// Function to calculate average time
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                    total_tat = 0;
// Function to find waiting time of all processes
      findWaitingTime(proc, n, wt);
// Function to find turn around time for all processes
```

```
    findTurnAroundTime(proc, n, wt, tat);
// Display processes along with all details
    cout << "Processes "
        << " Burst time "
        << " Waiting time "
        << " Turn around time\n";
 // Calculate total waiting time and total turnaround time
    for (int i = 0; i < n; i++) {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
            << proc[i].bt << "\t" << wt[i]
            << "\t\t " << tat[i] << endl;
    }
 cout << "\nAverage waiting time = "
        << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
        << (float)total_tat / (float)n;
}
 // Driver code
int main()
{
    Process proc[] = { { 1, 6, 1 }, { 2, 8, 1 },{ 3, 7, 2 }, { 4, 3, 3 } };
    int n = 4;
    findavgTime(proc, n);
    return 0;
}
```

❖ ***Priority (Non-preemptive) using C++:***

```
// C++ program for implementation of priority scheduling
#include<bits/stdc++.h>
using namespace std;
 struct Process
{
    int pid;     // Process ID
    int bt;      // CPU Burst time required
    int priority;  // Priority of this process
};
// Function to sort the Process acc. to priority
bool comparison(Process a, Process b)
{
    return (a.priority > b.priority);
}
// Function to find the waiting time for all processes
void findWaitingTime(Process proc[], int n,int wt[])
{
// waiting time for first process is 0
    wt[0] = 0;
// calculating waiting time
    for (int  i = 1; i < n ; i++ )
        wt[i] =  proc[i-1].bt + wt[i-1] ;
}
// Function to calculate turn around time
void findTurnAroundTime( Process proc[], int n,int wt[], int tat[])
{
```

```
// calculating turnaround time by adding bt[i] + wt[i]
    for (int  i = 0; i < n ; i++)
        tat[i] = proc[i].bt + wt[i];
}
//Function to calculate average time
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
//Function to find waiting time of all processes
    findWaitingTime(proc, n, wt);
//Function to find turn around time for all processes
    findTurnAroundTime(proc, n, wt, tat);
//Display processes along with all details
    cout << "\nProcesses  "<< " Burst time  "
         << " Waiting time  " << " Turn around time\n";
// Calculate total waiting time and total turn around time
    for (int  i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << "   " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t    " << wt[i]
             << "\t\t  " << tat[i] <<endl;
    }
    cout << "\nAverage waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
```

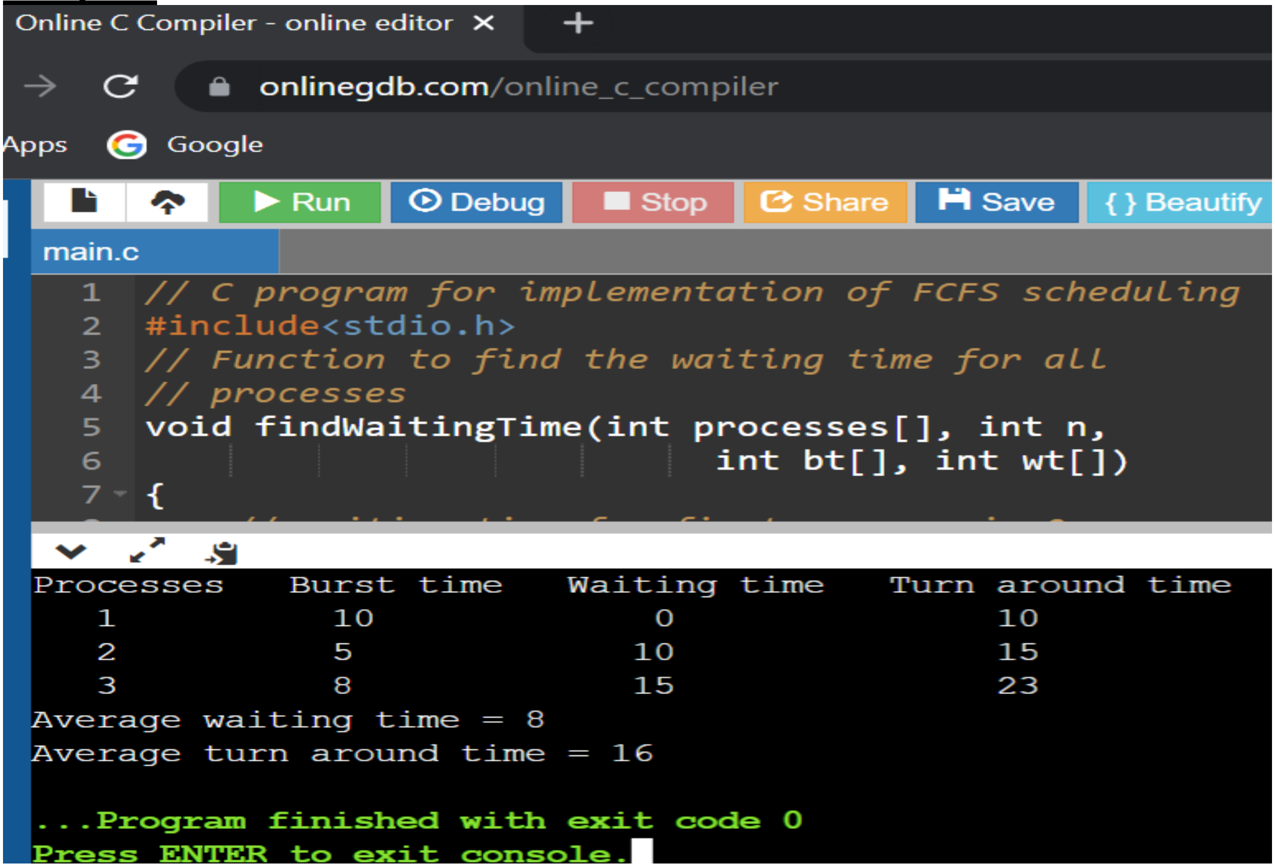
```
}
void priorityScheduling(Process proc[], int n)
{
    // Sort processes by priority
    sort(proc, proc + n, comparison);
cout<< "Order in which processes gets executed \n";
    for (int  i = 0 ; i <  n; i++)
        cout << proc[i].pid <<" " ;
   findavgTime(proc, n);
}
// Driver code
int main()
{
 Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};  //2 is the highest prioirty
    int n = 3;
    priorityScheduling(proc, n);
    return 0;
}
```

❖ ***Round Robin (Preemptive by default) using C++:***

```
// C++ program for implementation of RR scheduling
#include<iostream>
using namespace std;
// Function to find the waiting time for all processes
void findWaitingTime(int processes[], int n,
            int bt[], int wt[], int quantum)
{
 //Make a copy of burst times bt[] to store remaining burst times.
    int rem_bt[n];
    for (int i = 0 ; i < n ; i++)
        rem_bt[i] = bt[i];
        int t = 0;  // Current time
 // Keep traversing processes in round robin manner until all of them are
not done.
  while (1)
  {
    bool done = true;
 // Traverse all processes one by one repeatedly
    for (int i = 0 ; i < n; i++)
    {
//If burst time of a process is greater than then only need to process
further
      if (rem_bt[i] > 0)
      {
        done = false;    // There is a pending process

        if (rem_bt[i] > quantum)
        {
//Increase the value of t i.e. showshow much time a process has been
processed
          t += quantum;
//Decrease the burst_time of current process by quantum
          rem_bt[i] -= quantum;
        }

// If burst time is smaller than or equal to quantum. Last cycle for this
process
        else
        {
//Increase the value of t i.e. shows how much time a process has been
processed
          t = t + rem_bt[i];
//Waiting time is current time minus time used by this process
          wt[i] = t - bt[i];
//As the process gets fully executed make its remaining burst time = 0
          rem_bt[i] = 0;
        }
      }
    }
//If all processes are done
    if (done == true)
    break;
  }
}
```

```
//Function to calculate turn around time
void findTurnAroundTime(int processes[], int n,int bt[], int wt[], int tat[])
{
//calculating turnaround time by adding bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}
//Function to calculate average time
void findavgTime(int processes[], int n, int bt[],int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
//Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt, quantum);
// Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);
//Display processes along with all details
    cout << "Processes "<< " Burst time "
        << " Waiting time " << " Turn around time\n";
//Calculate total waiting time and total turn around time
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
            << wt[i] <<"\t\t " << tat[i] <<endl;
    }
   cout << "Average waiting time = "
        << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
        << (float)total_tat / (float)n;   }
// Driver code
int main()
{
// process id's
    int processes[] = { 1, 2, 3};
    int n = 3;
// Burst time of all processes
    int burst_time[] = {10, 5, 8};
 // Time quantum
    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}
```

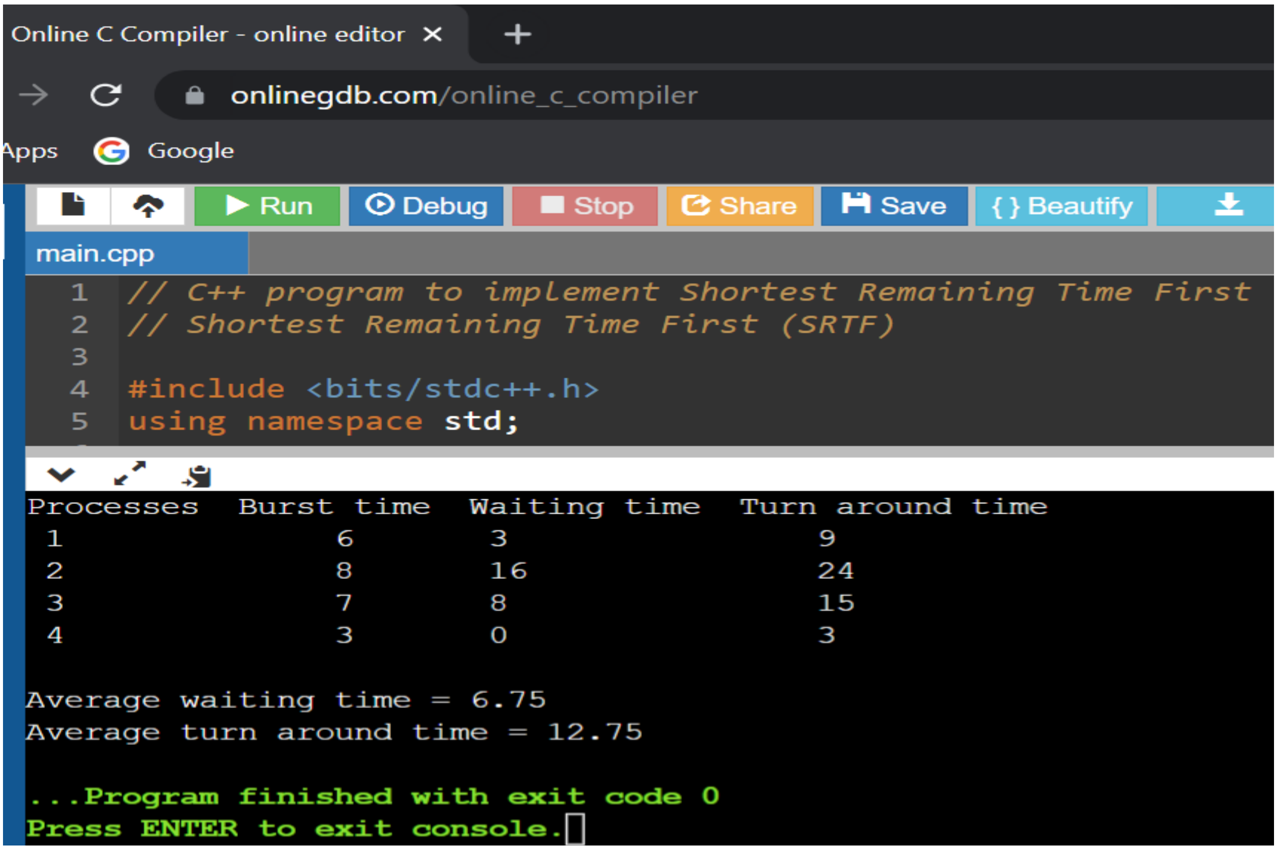
Output: FCFS:


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Share
Save
{ } Beautify
main.c

```
// C program for implementation of FCFS scheduling
#include<stdio.h>
// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
                          int bt[], int wt[])
{
```

```
Processes    Burst time    Waiting time    Turn around time
   1            10              0              10
   2            5               10             15
   3            8               15             23
Average waiting time = 8
Average turn around time = 16

...Program finished with exit code 0
Press ENTER to exit console.
```

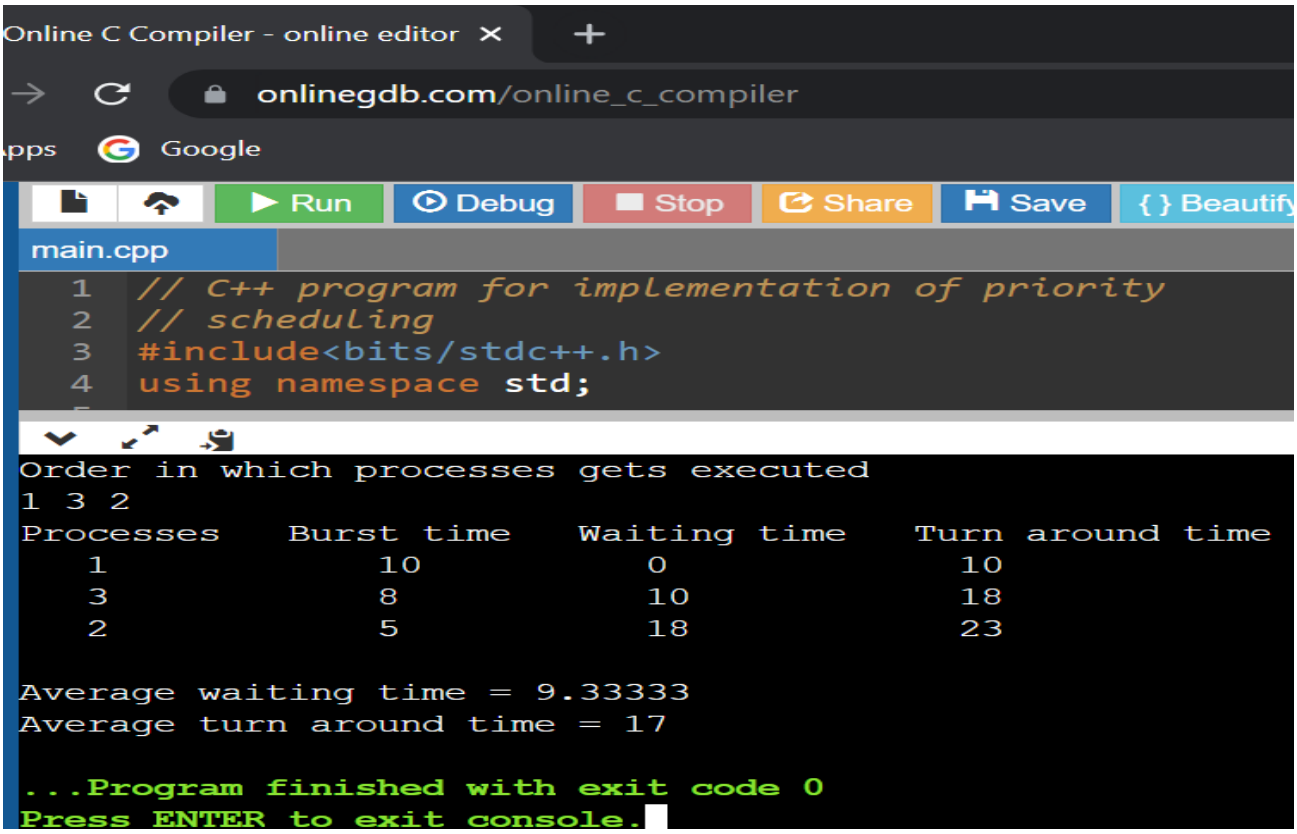
SRTF:


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Share
Save
{ } Beautify
main.cpp

```
// C++ program to implement Shortest Remaining Time First
// Shortest Remaining Time First (SRTF)

#include <bits/stdc++.h>
using namespace std;
```

```
Processes  Burst time  Waiting time  Turn around time
 1              6         3               9
 2              8         16              24
 3              7         8               15
 4              3         0               3

Average waiting time = 6.75
Average turn around time = 12.75

...Program finished with exit code 0
Press ENTER to exit console.
```

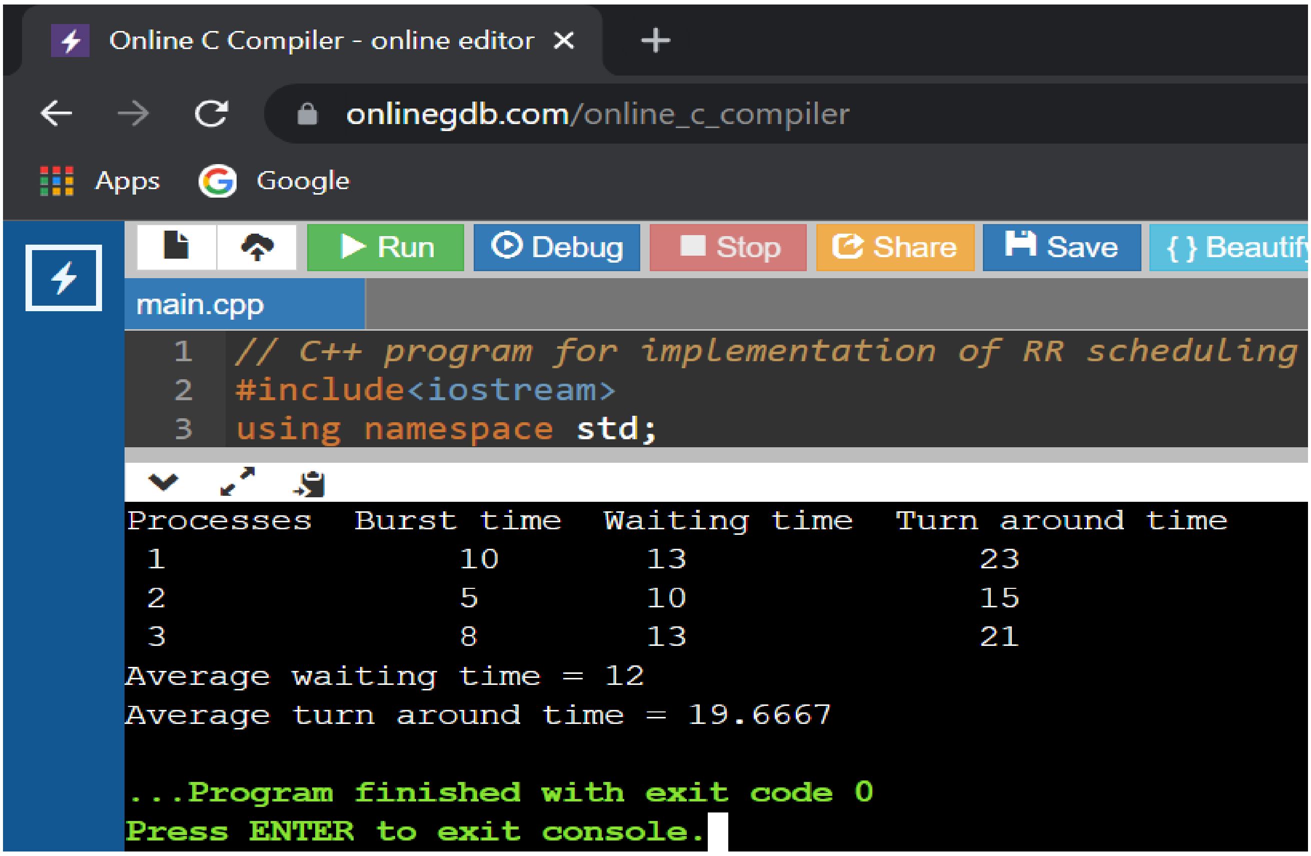
Priority (Non-preemptive) :


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
pps
Google
Run
Debug
Stop
Share
Save
{ } Beautify
main.cpp

```
// C++ program for implementation of priority
// scheduling
#include<bits/stdc++.h>
using namespace std;
```

```
Order in which processes gets executed
1 3 2
Processes    Burst time    Waiting time    Turn around time
   1            10            0              10
   3            8             10             18
   2            5             18             23

Average waiting time = 9.33333
Average turn around time = 17

...Program finished with exit code 0
Press ENTER to exit console.
```

Round Robin (Preemptive by default):


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Share
Save
{ } Beautify
main.cpp

```
// C++ program for implementation of RR scheduling
#include<iostream>
using namespace std;
```

```
Processes  Burst time  Waiting time  Turn around time
 1              10        13              23
 2              5         10              15
 3              8         13              21
Average waiting time = 12
Average turn around time = 19.6667

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion: We have Simulated the following scheduling algorithms FCFS using C language, SRTF using C++ language, Priority using C++ language and Round Robin using C++ language.

Part B

SPOS Lab Assignment 8

Assignment No: 8	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Banker's algorithm

Problem Statement: Simulation of Banker's algorithm using 'C'/Python/Java language.

Theory: **What is Banker's Algorithm?**

- Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.
- Used to **avoid deadlock** and **allocate resources** safely to each process in the computer system.
- The '**S-State'** examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes.

1. How much each process can request for each resource in the system. It is denoted by the [**MAX**] request.
2. How much each process is currently holding each resource in a system. It is denoted by the [**ALLOCATED**] resource.
3. It represents the number of each resource currently available in the system. It is denoted by the [**AVAILABLE**] resource.

Data structures:
Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available: It is a 1-d array of size **‘m’** indicating the number of available resources of each type.Available[j] = k means there are **‘k’** instances of resource type **R_j**

Max: It is a 2-d array of size ‘**n*m’** that defines the maximum demand of each process in a system.Max[i, j] = k means process **P_i** may request at most **‘k’** instances of resource type **R_j.**

Allocation: It is a 2-d array of size **‘n*m’** that defines the number of resources of each type currently allocated to each process.Allocation[i, j] = k means process **P_i** is currently allocated **‘k’** instances of resource type **R_j**

Need: It is a 2-d array of size **‘n*m’** that indicates the remaining resource need of each process.Need [i,   j] = k means process **P_i** currently need **‘k’** instances of resource type **R_j** for its execution.Need [i,   j] = Max [i,   j] – Allocation [i,   j]

Allocation specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

The Banker's Algorithm is the combination of the safety algorithm and the resource request algorithm to control the processes and avoid deadlock in a system:

Safety Algorithm: It is a safety algorithm used to check whether or not a system is in a safe state or follows the safe sequence in a banker's algorithm.

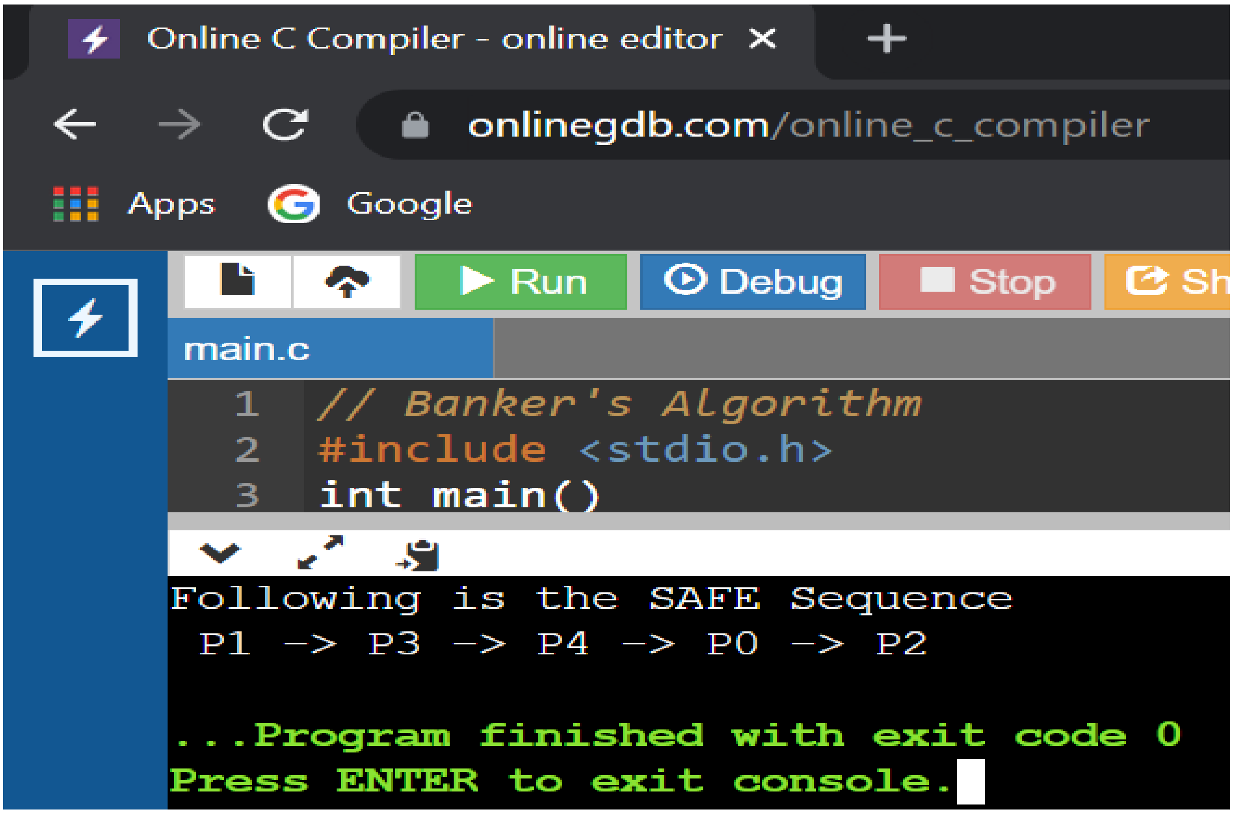
Resource Request Algorithm: A resource request algorithm checks how a system will behave when a process makes each type of resource request in a system as a request matrix.

Program:

```
// Banker's Algorithm

#include <stdio.h>

int main(){

 // P0, P1, P2, P3, P4 are the Process names here

  int n, m, i, j, k;
  n = 5;    // Number of processes
  m = 3;   // Number of resources
  int alloc[5][3] = { { 0, 1, 0 },    // P0    // Allocation Matrix
            { 2, 0, 0 },    // P1
            { 3, 0, 2 },   // P2
            { 2, 1, 1 },   // P3
            { 0, 0, 2 } };   // P4
 int max[5][3] = { { 7, 5, 3 },     // P0    // MAX Matrix
           { 3, 2, 2 },          // P1
           { 9, 0, 2 },          // P2
           { 2, 2, 2 },          // P3
           { 4, 3, 3 } };        // P4
int avail[3] = { 3, 3, 2 };          // Available Resources
  int f[n], ans[n], ind = 0;
  for (k = 0; k < n; k++) {
    f[k] = 0;  }
  int need[n][m];
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
      need[i][j] = max[i][j] - alloc[i][j]; }
  int y = 0;
  for (k = 0; k < 5; k++) {
    for (i = 0; i < n; i++) {
      if (f[i] == 0) {
       int flag = 0;
        for (j = 0; j < m; j++) {
          if (need[i][j] > avail[j]){
            flag = 1;
```

```
                         break;}}
  if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }}}}
 printf("Following is the SAFE Sequence\n");
   for (i = 0; i < n - 1; i++)
       printf(" P%d ->", ans[i]);
   printf(" P%d", ans[n - 1]);
return (0)  }
```

Output:


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Sh
main.c

```
// Banker's Algorithm
#include <stdio.h>
int main()
```

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion: We have Simulated/implemented Banker’s algorithm using C language.

Part B

SPOS Lab Assignment 9

Assignment No: 9	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Reader-writer process synchronization

Problem Statement: Implement the process synchronization with the structure of reader and writer process

Theory:

Consider a situation where we have a file shared between many people:

- If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
- However if some person is reading the file, then others may read it at the same time.
- Precisely in OS we call this situation as the **readers-writers problem**

Summary

- One set of data is shared among a number of processes
- Once a writer is ready, it performs its write. Only one writer may write at a time
- If a process is writing, no other process can read it
- If at least one reader is reading, no other process can write
- Readers may not write and only read together

Solution when the reader has the priority over writer

- Here priority means, no reader should wait if the share is currently opened for reading.
- Three variables are used: mutex, wrt, readcnt to implement solution
- semaphore mutex, wrt; // semaphore mutex is used to ensure mutual exclusion when readcnt is updated i.e. when any reader enters or exit from the critical section and semaphore wrt is used by both readers and writers
- int readcnt; // readcnt tells the number of processes performing read in the critical section, initially 0

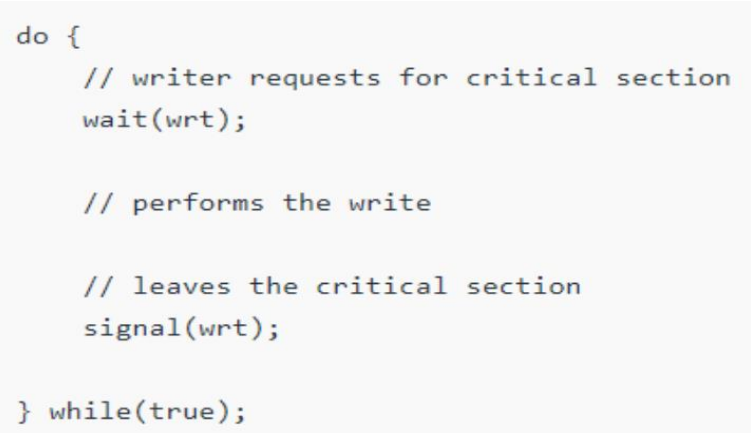
<u>Writer process:</u>

– Writer requests the entry to critical section.

– If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.

– It exits the critical section

Solution when the reader has the priority over writer:



```
do {
    // writer requests for critical section
    wait(wrt);

    // performs the write

    // leaves the critical section
    signal(wrt);

} while(true);
```

Solution when the reader has the priority over writer Reader process:

- Reader requests the entry to critical section.
- If allowed:

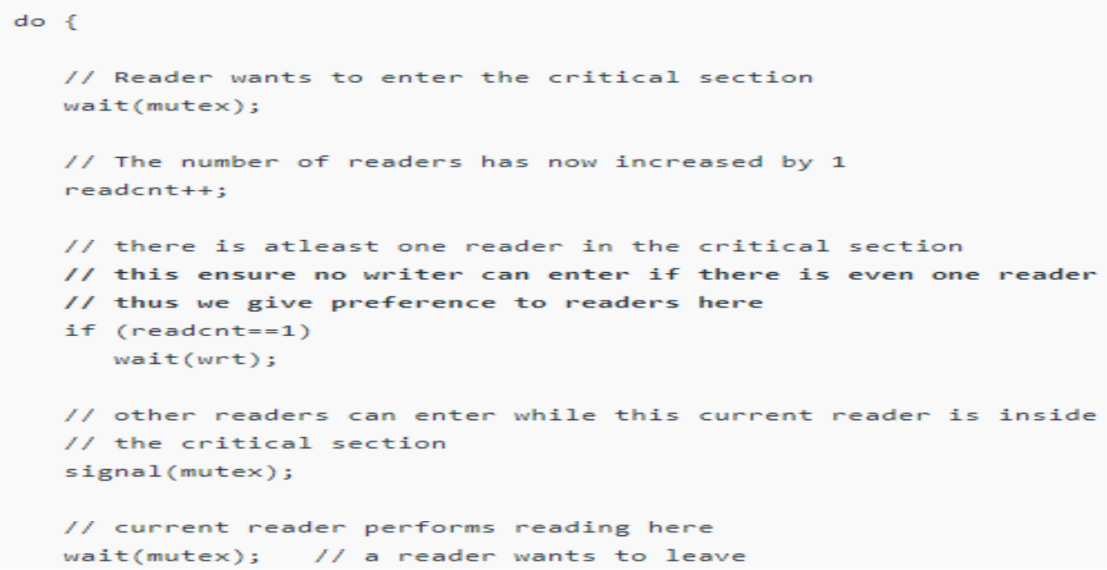
– it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.

– It then, signals mutex as any other reader is allowed to enter while others are already reading.

– After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wrt” as now, writer can enter the critical section.

- If not allowed, it keeps on waiting.

Thus, the semaphore ‘wrt‘ is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.



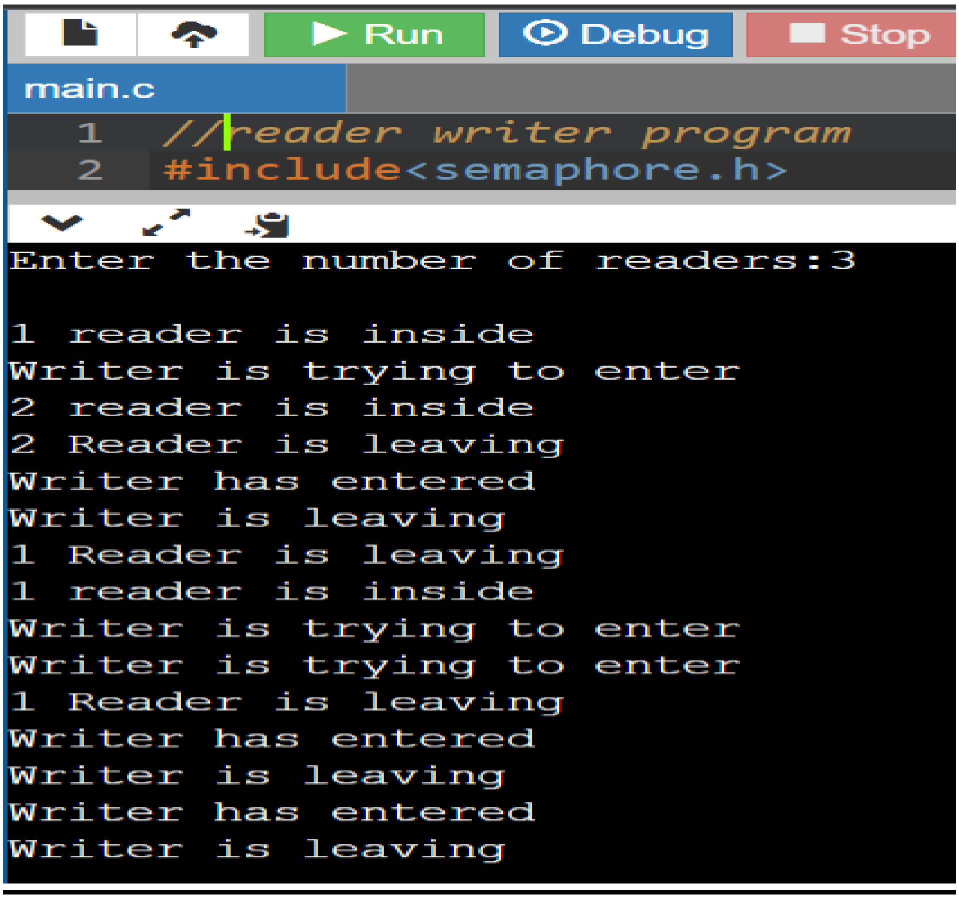
```
do {

   // Reader wants to enter the critical section
   wait(mutex);

   // The number of readers has now increased by 1
   readcnt++;

   // there is atleast one reader in the critical section
   // this ensure no writer can enter if there is even one reader
   // thus we give preference to readers here
   if (readcnt==1)
      wait(wrt);

   // other readers can enter while this current reader is inside
   // the critical section
   signal(mutex);

   // current reader performs reading here
   wait(mutex);   // a reader wants to leave
```

Program:

```
#include<semaphore.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

sem_t x,y;

pthread_t tid;

pthread_t writerthreads[100],readerthreads[100];

int readercount = 0;
```

```
void *reader(void* param)  {
    sem_wait(&x);
    readercount++;
if(readercount==1)
        sem_wait(&y);
    sem_post(&x);
    printf("%d reader is inside\n",readercount);
    usleep(3);
    sem_wait(&x);
    readercount--;
    if(readercount==0) {
        sem_post(&y); }
    sem_post(&x);
    printf("%d Reader is leaving\n",readercount+1);
    return NULL;
}
void *writer(void* param){
    printf("Writer is trying to enter\n");
    sem_wait(&y);
    printf("Writer has entered\n");
    sem_post(&y);
    printf("Writer is leaving\n");
    return NULL;
}
int main(){
    int n2,i;
    printf("Enter the number of readers:");
    scanf("%d",&n2);
    printf("\n");
    int n1[n2];
    sem_init(&x,0,1);
    sem_init(&y,0,1);
    for(i=0;i<n2;i++){
        pthread_create(&writerthreads[i],NULL,reader,NULL);
        pthread_create(&readerthreads[i],NULL,writer,NULL);}
    for(i=0;i<n2;i++){
        pthread_join(writerthreads[i],NULL);
        pthread_join(readerthreads[i],NULL);
    }}
```

Output:



```
//reader writer program
#include<semaphore.h>
```

```
Enter the number of readers:3

1 reader is inside
Writer is trying to enter
2 reader is inside
2 Reader is leaving
Writer has entered
Writer is leaving
1 Reader is leaving
1 reader is inside
Writer is trying to enter
Writer is trying to enter
1 Reader is leaving
Writer has entered
Writer is leaving
Writer has entered
Writer is leaving
```

Conclusion: We have implemented the process synchronization with the structure of reader and writer process.

Part B

SPOS Lab Assignment 10

Assignment No: 10	
Name:	Apekshita Mohan Kalbhor
Class:	TY CSE Core 1
Batch:	A
Roll No:	2193051
Enrollment No:	MITU19BTCS0133

Title: Page replacement algorithms

Problem Statement: Implement the following page replacement algorithms :

1. FIFO
2. LRU
3. Optimal

Theory: What is Page Replacement Algorithm?

This algorithm helps to decide which pages must be swapped out from the main memory in order to create a room for the incoming page.This Algorithm wants the lowest page-fault rate.

- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced.
- This approach is the Least Recently Used (LRU) algorithm. LRU replacement associates with each page the time of that page's last use.
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.

Types of Page Replacement Algorithms

1) FIFO: It is a very simple way of Page replacement and is referred to as First in First Out. This algorithm mainly replaces the oldest page that has been present in the main memory for the longest time.

- This algorithm is implemented by keeping the track of all the pages in the queue.
- As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim.
- This is not an effective way of page replacement but it can be used for small systems.

2) Optimal Page Replacement algorithm: This algorithm stands for "Least recent used" and this algorithm helps the Operating system to search those pages that are used over a short duration of time frame.

- The page that has not been used for the longest time in the main memory will be selected for replacement.
- This algorithm is easy to implement.
- This algorithm makes use of the counter along with the even-page.

3) Least recent used (LRU) page replacement algorithm: This algorithm mainly replaces the page that will not be used for the longest time in the future. The practical implementation of this algorithm is not possible.

- Practical implementation is not possible because we cannot predict in advance those pages that will not be used for the longest time in the future.
- This algorithm leads to less number of page faults and thus is the best-known algorithm Also, this algorithm can be used to measure the performance of other algorithms.

Data structures:
Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Program:

❖ *FIFO page replacement algorithm:*

```
# Python3 implementation of FIFO page replacement in Operating
Systems.
      from queue import Queue
# Function to find page faults using FIFO
     def pageFaults(pages, n, capacity):
# To represent set of current pages.We use an unordered_set so that
we quickly check if a page is present in set or not
    s = set()
# To store the pages in FIFO manner
    indexes = Queue()
# Start from initial page
     page_faults = 0
    for i in range(n):
# Check if the set can hold more pages
        if (len(s) < capacity):
# Insert it into set if not present already which represents page fault
            if (pages[i] not in s):
                s.add(pages[i])
# increment page fault
                page_faults += 1
# Push the current page into the queue
                indexes.put(pages[i])
# If the set is full then need to perform FIFO i.e. remove the first page of
the queue from set and queue both and insert the current page
        else:
 # Check if current page is not already present in the set
```

```
            if (pages[i] not in s):
   # Pop the first page from the queue
                val = indexes.queue[0]
                indexes.get()
  # Remove the indexes page
                s.remove(val)
  # insert the current page
                s.add(pages[i])
# push the current page into the queue
                indexes.put(pages[i])
# Increment page faults
                page_faults += 1
  return page_faults
# Driver code
if __name__ == '__main__':
    pages = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]
    n = len(pages)
    capacity = 4
    print(pageFaults(pages, n, capacity))
```

❖ LRU page replacement algorithm:

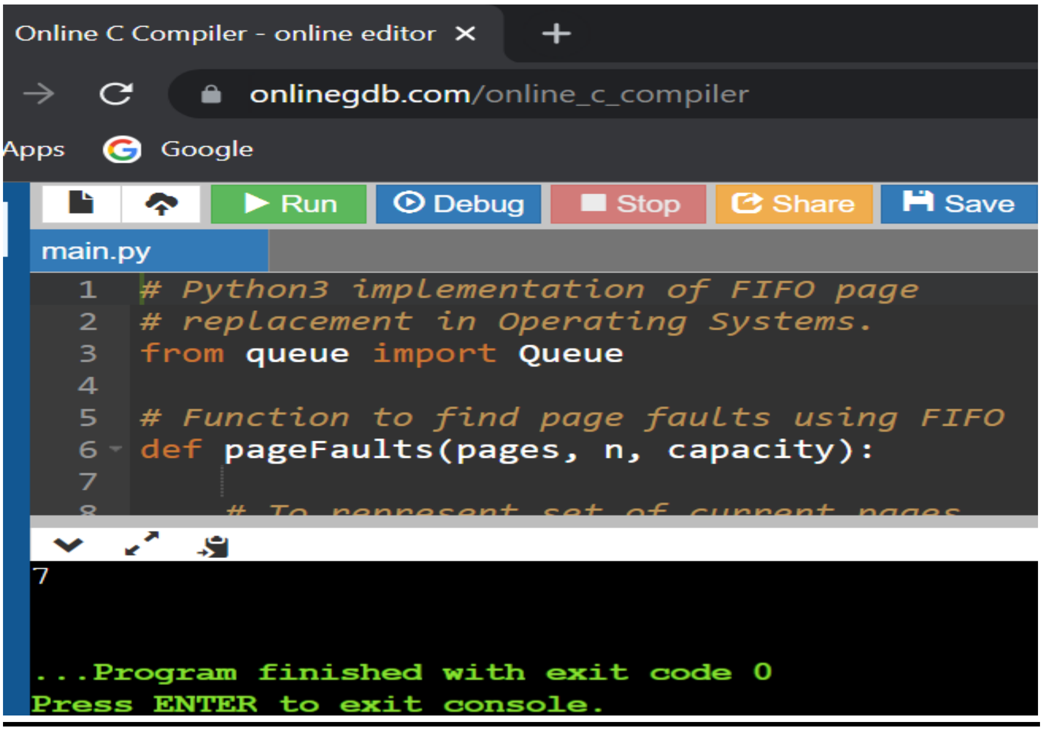
```
# LRU page replacement Algorithm
print("Enter the number of frames: ",end="")
capacity = int(input())
f,st,fault,pf = [],[],0,'No'
print("Enter the reference string: ",end="")
s = list(map(int,input().strip().split()))
```

```
print("\nString|Frame →\t",end='')
for i in range(capacity):
    print(i,end=' ')
print("Fault\n   ↓\n")
for i in s:
    if i not in f:
        if len(f)<capacity:
            f.append(i)
            st.append(len(f)-1)
        else:
            ind = st.pop(0)
            f[ind] = i
            st.append(ind)
        pf = 'Yes'
        fault += 1
    else:
        st.append(st.pop(st.index(f.index(i))))
        pf = 'No'
    print("   %d\t\t"%i,end='')
    for x in f:
        print(x,end=' ')
    for x in range(capacity-len(f)):
        print(' ',end=' ')
    print(" %s"%pf)
print("\nTotal Requests: %d\nTotal Page Faults: %d\nFault Rate: %0.2f%%"%(len(s),fault,(fault/len(s))*100))
```

Optimal page replacement algorithm:

```
// CPP program to demonstrate optimal page replacement algorithm.
#include <bits/stdc++.h>
using namespace std;
// Function to check whether a page exists in a frame or not
bool search(int key, vector<int>& fr)
{
    for (int i = 0; i < fr.size(); i++)
        if (fr[i] == key)
            return true;
    return false;
}
//Function to find the frame that will not be usedrecently in future after
given index in pg[0..pn-1]
int predict(int pg[], vector<int>& fr, int pn, int index)
{
//Store the index of pages which are going to be used recently in future
    int res = -1, farthest = index;
    for (int i = 0; i < fr.size(); i++) {
        int j;
        for (j = index; j < pn; j++) {
            if (fr[i] == pg[j]) {
                if (j > farthest) {
                    farthest = j;
                    res = i;
                }
                break;
            }}
//If a page is never referenced in future,return it.
        if (j == pn)
            return i;
    }
// If all of the frames were not in future,return any of them, we return 0.
Otherwise we return res.
    return (res == -1) ? 0 : res;
}
void optimalPage(int pg[], int pn, int fn)
{
// Create an array for given number of frames and initialize it as empty.
    vector<int> fr;
//Traverse through page reference array and check for miss and hit.
    int hit = 0;
    for (int i = 0; i < pn; i++) {
// Page found in a frame : HIT
        if (search(pg[i], fr)) {
            hit++;
            continue;
        }
// Page not found in a frame : MISS  ,If there is space available in
frames.
        if (fr.size() < fn)
            fr.push_back(pg[i]);
// Find the page to be replaced.
        else {
            int j = predict(pg, fr, pn, i + 1);
            fr[j] = pg[i];} }
    cout << "No. of hits = " << hit << endl;
    cout << "No. of misses = " << pn - hit << endl;  }
// Driver Function
int main()  {
    int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };
    int pn = 13;
    int fn = 4;
    optimalPage(pg, pn, fn);
    return 0;  }
```

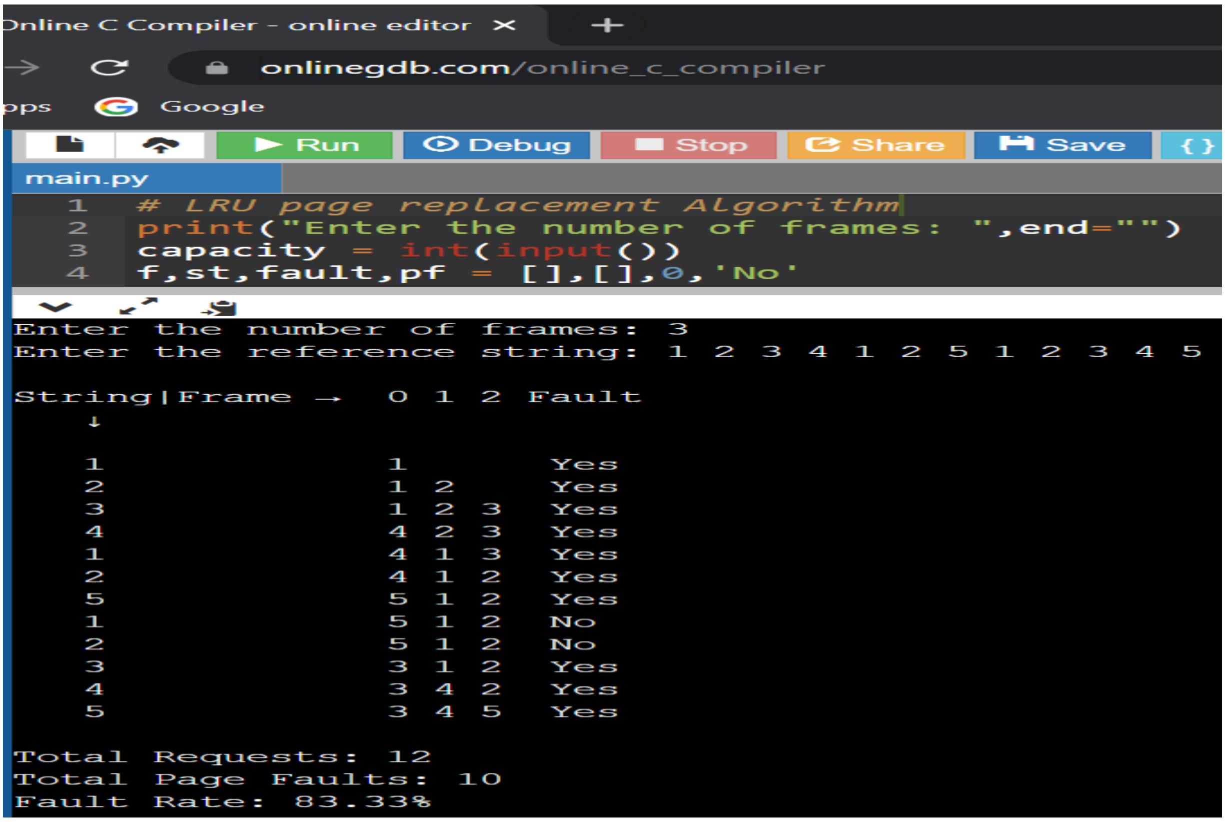
Output: FIFO:


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Share
Save
main.py

```
# Python3 implementation of FIFO page
# replacement in Operating Systems.
from queue import Queue

# Function to find page faults using FIFO
def pageFaults(pages, n, capacity):

    # To represent set of current pages
```

```
7


...Program finished with exit code 0
Press ENTER to exit console.
```

LRU:

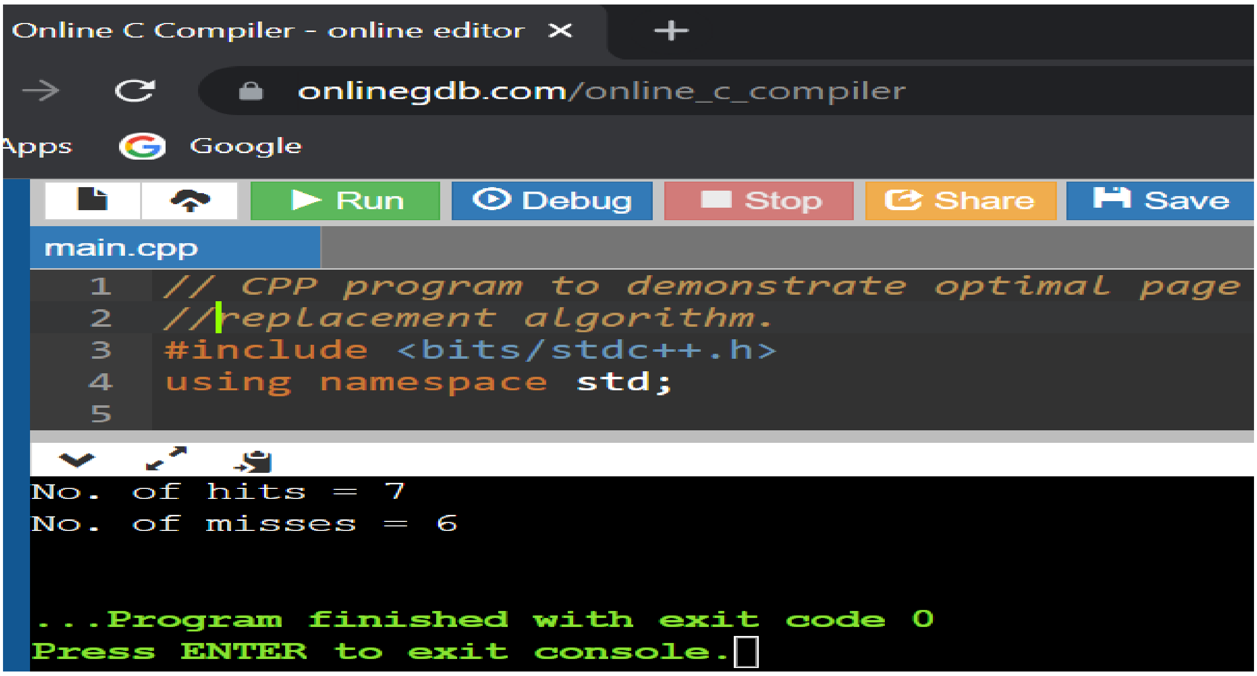


nline C Compiler - online editor
onlinegdb.com/online_c_compiler
pps
Google
Run
Debug
Stop
Share
Save
{ }
main.py

```
# LRU page replacement Algorithm
print("Enter the number of frames: ",end="")
capacity = int(input())
f,st,fault,pf = [],[],0,'No'
```

```
Enter the number of frames: 3
Enter the reference string: 1 2 3 4 1 2 5 1 2 3 4 5

String|Frame →   0 1 2 Fault
   ↓

   1             1       Yes
   2             1 2     Yes
   3             1 2 3   Yes
   4             4 2 3   Yes
   1             4 1 3   Yes
   2             4 1 2   Yes
   5             5 1 2   Yes
   1             5 1 2   No
   2             5 1 2   No
   3             3 1 2   Yes
   4             3 4 2   Yes
   5             3 4 5   Yes

Total Requests: 12
Total Page Faults: 10
Fault Rate: 83.33%
```

Optimal:



Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Apps
Google
Run
Debug
Stop
Share
Save
main.cpp

```
// CPP program to demonstrate optimal page
//replacement algorithm.
#include <bits/stdc++.h>
using namespace std;

```

```
No. of hits = 7
No. of misses = 6


...Program finished with exit code 0
Press ENTER to exit console.
```

<u>Conclusion:</u> We have implemented the following page replacement algorithms : FIFO, LRU in python language and Optimal in C++ language.